UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS GRADUAÇÃO EM INFORMÁTICA

ENDHE ELIAS SOARES

JOINT-E: UM FRAMEWORK PARA AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE DE APIS DE PERSISTÊNCIA EM ONTOLOGIAS

Maceió
23/05/2014

Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecário: Roselito de Oliveira Santos

S676j Soares, Endhe Elias.
Joint-e: um framework para avaliação de desempenho e escalabilidade de apis de persistência em ontologias / Endhe Elias Soares. – Maceió, 2014.
83 f. : il.

Orientador: Ig Ibert Bittencourt Santana Pinto.
Coorientador: Seiji Isotani.
Dissertação (Mestrado em Informática) – Universidade Federal de Alagoas. Instituto de Computação. Programa de Pós-Graduação em Informática. Maceió, 2014.

Bibliografia: f. 67-70.

1. Desempenho de softwares . 2.Escalabilidade de softwares 3.Framework. I. Título.

CDU: 004.45

UNIVERSIDADE FEDERAL DE ALAGOAS/UFAL
Programa de Pós-Graduação em Informática – PpgI
Instituto de Computação
Campus A. C. Simões BR 104-Norte Km 14 BL 12 Tabuleiro do Martins
Maceió/AL - Brasil CEP: 57.072-970 | Telefone: (082) 3214-1401

Membros da Comissão Julgadora da Dissertação de Mestrado de Endhe Elias Soares, intitulada: "JOINT-E: Um Framework para Avaliação de Desempenho e Escalabilidade de APIs de Persistência em Ontologias", apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Alagoas em 23 de maio de 2014, às 08h00min, no miniauditório do Instituto de Computação da UFAL.

COMISSÃO JULGADORA

Prof. Dr. Ig Ibert Bittencourt Santana Pinto
UFAL – Instituto de Computação
Orientador

Prof. Dr. Seiji Isotani
ICMC - USP
Coorientador

Prof. Dr. Leandro Dias da Silva
Instituto de Computação – UFAL
Examinador

Prof. Dr. Márcio de Medeiros Ribeiro
Instituto de Computação – UFAL
Examinador

Prof. Dr. Clóvis Torres Fernandes
Instituto Tecnológico de Aeronáutica - ITA
Examinador

Endhe Elias Soares

JOINT-E: UM FRAMEWORK PARA AVALIAÇÃO DE DESEMPENHO E ESCALABILIDADE DE APIS DE PERSISTÊNCIA EM ONTOLOGIAS

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal de Alagoas.
Orientador: Prof. Dr. Ig Ibert Bittencourt
Coorientador: Prof. Dr. Seiji Isotani

Maceió
23/05/2014

Dedico este trabalho as luzes da minha vida, meu filho Pedro Lucca, minha linda esposa Juliana, também aos meus pais e avós pelo apoio em todo caminho.

AGRADECIMENTOS

Tudo somente é possível como o consentimento de Deus e a ele que faço meu primeiro agradecimento.

Gostaria de agradecer também aos meus pais, Eduardo e Josiene, pelo apoio em todo caminho, que quase sempre é cheio de obstáculos, mas com eles a jornada tornasse melhor.

Agradeço aos meus avôs, Jonas Elias (in memoriam), Maria Lúcia, Manoel Soares (in memoriam) e Isaura Soares (in memoriam). A interrupção do convívio deixa saudades, mas sinto que estão comigo, olhando e cuidando.

Não poderia deixar de agradecer a Juliana, hoje minha esposa e meu eterno amor, por toda dedicação, respeito e carinho que vem me dando nesses anos de convivência. Agradeço também ao meu filho Pedro Lucca, por me mostrar o amor incondicional.

Agradecimento especial aos meus amigos Ig Bittencourt e Olavo Holanda, além de contribuir significativamente com este trabalho, tornaram-se meus grandes amigos.

À todos da família NEES, Diego Dermeval, Judson Bandeira, Williams Alcântara (Todynho), Armando e demais membros.

Também agradeço a minha segunda casa, MeuTutor, meus amigos, Alan Pedro, Marcelo Nunes, Thyago Tenório, Seiji Isotani, Wilkson Eldon, Heiner e tantos outros, meu muito obrigado.

RESUMO

A Web Semântica (WS) vem se tornando um importante tópico de pesquisa na Ciência da Computação. Uma das razões é a possibilidade de representar a informação de maneira semântica por meio de ontologias. Como consequência, inúmeras aplicações têm sido desenvolvidas utilizando as tecnologias da Web semântica, tais como, RDF, SPARQL e as próprias ontologias. Embora o desenvolvimento de software utilizando estas tecnologias seja complicado e custoso, a comunidade da WS vem produzindo ferramentas e APIs (*application programming interface*) para apoiar programadores no desenvolvimento de aplicações semânticas.

Nesse cenário, existem atualmente duas principais abordagens utilizadas por essas APIs para manipulação de ontologias: triplas RDF e Programação Orientada a Objetos (POO). Por um lado, o uso de APIs para manipular triplas RDF permite que o desenvolver crie aplicações mapeando as propriedades das classes presentes nas triplas RDF em código utilizando uma linguagem de programação (e.g. Java). Por outro lado, existem as APIs para manipular ontologias utilizando o paradigma de orientação a objetivo. Isso permite que desenvolvedores continuem utilizando um paradigma já conhecido e largamento utilizado. Embora várias APIs tenham sido desenvolvidas para manipular ontologias no nível de objeto, a maioria não foi adequadamente avaliada, principalmente no que se refere aos atributos de qualidade de desempenho e escalabilidade. Além disso, a alta quantidade e a variabilidade das APIs faz com que seja necessário construir uma abordagem genérica que lide com essas questões, uma vez que construir um sistema de avaliação para cada API isoladamente é inviável.

Dessa forma, este trabalho apresenta um framework centrado na arquitetura, chamado JOINT-E (Java Ontology Integration Toolkit-Evaluator), que permite a desenvolvedores avaliar as APIs para manipulação de ontologias. Para realizar esta avaliação foi objetivo desta trabalho definir um conjunto de métricas de desempenho e escalabilidade. Frisa-se que o framework e as métricas definidas possibilitam a análise e comparação das APIs com apoio estatístico, aumentando a credibilidade e confiabilidade dos resultados.

Para validar os resultados obtidos foi proposto um experimento com três cenários usando as principais APIs (Alibaba e Jastor) utilizadas pela comunidade de desenvolvimento de aplicativos semântico. De acordo com este experimento a Jastor apresentou melhor performance em relação ao Alibaba levando em consideração as métricas propostas. Por fim, realizou-se também uma pesquisa de opinião com desenvolvedores para verificar se o framework JOINT-E oferece informações importantes para tomada de decisão referente a escolha de uma API. Os resultados desta pesquisa constaram que o framework oferece meios mais precisos para avaliação das APIs e atende a demanda dos desenvolvedores.

Palavras-chaves: Desempenho. Escalabilidade. Framework.

ABSTRACT

The Semantic Web (WS) is becoming an important research topic in Computer Science. One of reasons is the possibility of representing information in semantic way using ontologies, assisting in the building of applications which are able to use data, becoming more scalable and intelligent. As a result, many new applications have been developed using the Semantic Web technologies such as RDF, SPARQL and ontologies themselves. Although software development using these technologies is complicated and costly, the community of WS has been producing tools and APIs (*application programming interface*) to support programmers in the development of semantic applications. Among the most important APIs developed by the community are those that provide mechanisms for handling ontologies.

Currently, there are two main approaches used by manipulation APIs: i) RDF triples and object-oriented programming (OOP). On the one hand, using the RDF triples, the developers have to manipulate the ontologies using only triples, making the development process more complicated. On the other hand, the OOP APIs promote ontologies manipulation using object, which facilitates the development.

Although several APIs are been developed in order to manipulate ontologies at object level, most of them are not being evaluated, especially when related to the quality of attributes, such as performance and scalability. Moreover, the high quantity and variability of APIs require a more generic approach in order to deal with these issues, because building an evaluation system for each API is a costly task and does not allow the reuse of the solution, neither of its modules.

Therefore, this work presents an architecture-centered framework, named JOINT-Evaluator (JOINT-E), where the developers will be able to evaluate the APIs based on a set of predefined performance and scalability metrics. It is important to note that the framework also enables the analysis and comparison of the data with statistical support, increasing the credibility and reliability of the results.

To validate the work three scenarios were created on experiment. The main APIs (Alibaba and Jastor) used by developers were tested and evaluated. Furthermore, a qualitative analysis is presented, showing the statistical results and highlighting the advantages and disadvantages of each API. The results have showed that Jastor surpass Alibaba in many issues. Finally, a survey was applied to developers in order to validate the framework. This suvery have presented that this work attend a developers demand.

Keywords: Performance. Scalability. Framework.

LISTA DE ILUSTRAÇÕES

Figura 1 – As camadas da Web Semântica 16
Figura 2 – Ontologia SIOC 18
Figura 3 – Compartilhamento e integração de ontologias 18
Figura 4 – Exemplo de código da ferramenta Sesame 19
Figura 5 – Exemplo de código da ferramenta Alibaba 20
Figura 6 – Metodologia GQM (*Goal-Question-Metric*) 20
Figura 7 – Exemplo de Componente COSMOS* 25
Figura 8 – Visão Geral do JOINT-E 33
Figura 9 – Diagrama de *Features* 35
Figura 10 – Diagrama de Casos de Uso 36
Figura 11 – Diagrama de Componentes 40
Figura 12 – Interface Gráfica do JOINT-E 41
Figura 13 – Componente - *Performance* 44
Figura 14 – Fluxo de execução geral do JOINT-E 45
Figura 15 – Cenário de configuração do Alibaba 49
Figura 16 – Seleção de features utilizando a API Alibaba 49
Figura 17 – Distribuição de valores referentes as operações *create* e *retrieve* da API Alibaba - Feature *Responsiveness* 50
Figura 18 – Distribuição de valores referentes as operações *update* e *delete* da API Alibaba - Feature *Responsiveness* 51
Figura 19 – Box-plots referentes as operações *create* e *retrieve* da API Alibaba - Feature *Resource Utilization* 51
Figura 20 – Box-plots referentes as operações *update* e *delete* da API Alibaba - Feature *Resource Utilization* 52
Figura 21 – Box-plot referente a Escalabilidade da Alibaba - Feature *Simultaneous Requests* 52
Figura 22 – Cenário de configuração da API Jastor 53
Figura 23 – Cenário de configuração da API Jastor 54
Figura 24 – Box-plots referentes as operações *create* e *retrieve* da API Jastor - Feature *Responsiveness* 54
Figura 25 – Box-plots referentes as operações *update* e *delete* da API Jastor - Feature *Responsiveness* 55
Figura 26 – Box-plots referentes as operações *create* e *retrieve* da API Jastor - Feature *Resource Utilization* 55
Figura 27 – Box-plots referentes as operações *update* e *delete* da API Jastor - Feature *Resource Utilization* 56

Figura 28 – Box-plot referente a Escalabilidade da Jastor - Feature *Simultaneous Requests* 56
Figura 29 – Comparação entre as capacidades de respostas das APIs Alibaba e Jastor 57
Figura 30 – Comparação entre consumo de memória das APIs Alibaba e Jastor . . 58
Figura 31 – Comparação entre consumo de memória das APIs Alibaba e Jastor . . 59
Figura 32 – Estados dos partipantes 60
Figura 33 – Nível de formação dos partipantes 60
Figura 34 – Atuação na pesquisa científica 60
Figura 35 – Atuação no mercado de trabalho 61
Figura 36 – Linguagens utilizadas 61
Figura 37 – Experiência de programação 61
Figura 38 – Frequência de avaliação 62
Figura 39 – Aspectos das APIs 62
Figura 40 – Necessidade de avaliação de desempenho relacionada ao tempo de resposta 63
Figura 41 – Necessidade de avaliação de desempenho relacionada à utilização de recurso 63
Figura 42 – Aspectos da avaliação de Escalabilidade 63
Figura 43 – Necessidade de um mecanismo de avaliação 64
Figura 44 – Tipos de visualização de dados 64
Figura 45 – Necessidade de um mecanismo de comparação 65
Figura 46 – Diagrama de Pacotes - Componente *Operações* 72

SUMÁRIO

1 INTRODUÇÃO 10
1.1 Escopo 12
1.2 Objetivos 13
1.3 Organização do texto 13

2 EMBASAMENTO TEÓRICO E TRABALHOS RELACIONADOS 15
2.1 Web Semântica 15
2.2 Ontologias 17
2.3 Desenvolvimento de Aplicações baseadas em Ontologias 18
2.3.1 Desenvolvimento orientado a triplas RDF 19
2.3.2 Desenvolvimento orientado a objetos 19
2.4 Metodologia *Goal-Question-Metric* 20
2.5 *Framework* 21
2.6 Desenvolvimento Centrado na Arquitetura 22
2.7 Desenvolvimento Baseado em Componentes 22
2.7.1 Metodologia COSMOS* 24
2.8 Trabalhos Relacionados 25

3 JOINT-E 27
3.1 Planos GQM 27
3.1.1 Plano GQM - Desempenho 28
3.1.2 Plano GQM - Escalabilidade 30
3.2 Análise de dados 31
3.2.1 Hipóteses para o Desempenho 32
3.2.2 Hipóteses para o Escalabilidade 32
3.3 Visão Geral 33
3.4 Projeto Arquitetural 34
3.4.1 Modelo de Features 35
3.4.2 Casos de Uso 35
3.4.3 Diagrama de Componentes 38
3.5 GUI 41
3.6 Serviços Gerais 41
3.7 Serviços de Aplicação 42
3.8 Implementação 43
3.9 Fluxo de Execução 45
3.10 Ferramentas e Persistência 45

3.11 Como instanciar o JOINT-E 46
3.12 Diretrizes do JOINT-E 46

4 EXPERIMENTO 48
4.1 Cenários 48
4.1.1 Cenário 1: API Alibaba 48
4.1.2 Cenário 2 52
4.1.3 Cenário 3 56
4.1.4 Discussão sobre os Cenários 59
4.2 Pesquisa de Opinião 59
4.2.1 Perfil dos Usuários 59
4.2.2 Aspectos de Avaliação 62

5 CONCLUSÃO 66

Referências 67

APÊNDICE A – INSTANCIANDO O JOINT-E 71

APÊNDICE B – CÓDIGOS 74
B.1 Instanciação do JOINT-E usando a Alibaba 74
B.2 Arquivo XML 78

APÊNDICE C – QUESTIONÁRIO 82

1 INTRODUÇÃO

Nos últimos anos, a Web Semântica vem se tornando um importante tópico de pesquisa na ciência da computação, onde a mesma tem sido reconhecida tanto na academia quanto na indústria (HARMELEN, 2006; TERZIYAN; KONONENKO, 2003). Contudo, de acordo com vários pesquisadores e especialistas em engenharia de software, o desenvolvimento de software utilizando a perspectiva da Web Semântica ainda é complicado e demorado, uma vez que as equipes de desenvolvimento precisam lidar com vários aspectos simultâneos, tais como arquiteturas de referência, padrões de projetos, metodologias de desenvolvimento, dentre outros (CARDOSO; LYTRAS, 2009; HENDLER et al., 2008). Frisa-se que esses aspectos, em sua maioria, encontram-se em estágio inicial e não validados sob a perspectiva da Web Semântica.

Para sanar este problema, a comunidade internacional da Web Semântica vem produzindo várias ferramentas de suporte para as equipes de desenvolvimento. Dentre elas, as que possuem grande destaque são as APIs (Application programming interface). De acordo com o World Wide Web Consortium (W3C), essas APIs podem ser categorizadas[1], como armazenamento em triplas, raciocinadores em ontologias, manipulação de ontologias, dentre outras. De fato, os conjuntos de APIs disponíveis em diferentes categorias tem potencial para oferecem o apoio desejado para criar aplicações Semânticas. Ressalta-se que uma das categorias mais difundidas é de APIs de manipulação de ontologias, que constam com uma lista de mais de 70 soluções para apoiar as equipes de desenvolvimento durante a criação de aplicações baseadas na Web semântica.

Estas APIs de manipulação de ontologias oferecem supporte ao processo de desenvolvimento, auxiliando no mapeamento e manipulação da ontologia no código da aplicação. Os principais paradigmas seguidos por elas são: triplas RDF (Resource Description Framework [2]) e o paradigma POO (Programação Orientada a Objetos). Cada paradigma possui suas vantagens e desvantagens como discutiremos a seguir. A tecnologia do RDF foi uma das precursoras da Web Semântica, sendo uma característica principal a estrutura de tripla, composta por: sujeito, predicado e objeto. Quando as equipes de desenvolvimento estão utilizando as APIs pertencentes a esse paradigma, deve-se manipular as ontologias utilizando triplas RDF. Isso permite criar aplicações semânticas com alto grau de complexidade, pois trabalha-se diretamente com o grafo RDF de uma ontologia. Apesar deste benefício, um dos grandes problemas do uso deste paradigma é a falta de um mapeamento transparente entre o código da aplicação e a ontologia. Por exemplo, caso um desenvolvedor deseje adicionar em uma ontologia uma classe com várias propriedades inerentes a

[1] Mais detalhes em: <http://www.w3.org/2001/sw/wiki/Main_Page>. Último acesso: Fevereiro de 2014

[2] Seu documento base publicado em 2004, estando assim disponível em: <http://www.w3.org/TR/rdf-primer/>

ela, faz-se necessário escrever várias linhas de código representando cada tripla da classe, que captura apenas um valor para cada propriedade. Similarmente, se o desenvolvedor desejar remover esta classe, várias linhas de código referentes às triplas das propriedades também devem ser escritas. Por outro lado, os desenvolvedores que utilizam as APIs do paradigma POO manipulam a ontologia no nível do objeto. Dessa maneira, essas APIs proveem uma correlação direta entre a ontologia e o código da aplicação, permitindo assim que o desenvolvimento seja mais simples e flexível. Por exemplo, uma classe no código representa uma classe na ontologia, sendo seus atributos mapeados como propriedades na ontologia. Quando o desenvolvedor deseja adicionar na ontologia uma classe com várias propriedades, basta adicionar sua classe equivalente no código da aplicação, facilitando assim o desenvolvimento. O problema do uso deste paradigma é que ele dificulta o entendimento da ontologia, por parte do desenvolvedor, potencializando o uso inadequado da ontologia (e os conceitos contidos nela) para desenvolver aplicativos.

As APIs do paradigma POO proveem uma solução sofisticada para o mapeamento de ontologia em objetos, facilitando assim o uso de ontologias para desenvolver e manter aplicativos semânticos (WENZEL, 2010). Dessa maneira, vários desenvolvedores migraram suas APIs, anteriormente no paradigma de triplas RDF para POO.

Atualmente, existem várias APIs que permitem aos desenvolvedores manipular a ontologia no nível de objeto, tais como: Alibaba (LEIGH, 2011), Jastor (SZEKELY; BETZ, 2009), OWL2Java (ZIMMERMANN, 2014), Empire (GROVE, 2010), dentre outras. Cada API é caracterizada por qualidades únicas, como desempenho, escalabilidade, suporte a consultas, flexibilidade e facilidade na integração. Assim, para desenvolver adequadamente uma aplicação é necessário mapear os requisitos da aplicação API (e.g. desempenho, escalabilidade, robustez, segurança) como as características e qualidades das APIs. Por exemplo, se a aplicação em desenvolvimento for executada em ambientes com restrição de recursos de máquina, a API escolhida deve consumir minimamente esses recursos (e.g. memória e CPU). Sendo assim, é necessário que o mecanismo de avaliação forneça aos desenvolvedores não somente precisão nos resultados obtidos, mas também a possibilidade de comparar os resultados provenientes das avaliações das APIs.

Devido à quantidade de APIs existentes e a falta de métricas bem definidas para avaliá-las adequadamente, existe uma grande demanda por parte da comunidade para a construção de uma abordagem de avaliação e comparação desta APIs. Esta Abordagem precisa ser genérica o suficiente para a lidar com o alto número de soluções disponíveis. Além disso, a construção de uma abordagem focada no reuso, traz vários benefícios para os desenvolvedores, tais como: maior confiabilidade, redução de riscos do processo e principalmente desenvolvimento acelerado (SOMMERVILLE, 2004). Dessa maneira, uma abordagem plausível é o uso de framework, fazendo com que haja não apenas o reuso do código, mas também o reuso do projeto de software (JOHNSON; FOOTE, 1988). É importante destacar que o entendimento do código do framework, bem como seus artefatos

arquiteturais, devem ser compreendidos com o mínimo de esforço por parte dos desenvolvedores. Dessa maneira, os módulos do framework precisam estar bem definidos, com suas responsabilidades e dependências explícitas. Todavia, fatores como produtividade e qualidade no processo de construção do framework também devem ser considerados. Portanto, uma estratégia de implantação viável é o desenvolvimento centrado na arquitetura (GARLAN; SHAW, 1994) junto com o desenvolvimento baseado em componentes (SZYPERSKI, 1998).

Neste contexto, a presente dissertação aborda o problema de avaliação das APIs POO de manipulação de ontologias por parte dos desenvolvedores, propondo um conjunto de critérios/métricas de desempenho e escalabilidade para auxiliar os desenvolvedores no processo de avaliação, análise e comparação de APIs. Propõem-se também um framework centrado na arquitetura, chamando JOINT- Evaluator (JOINT[3]-E), no qual os desenvolvedores são capazes de avaliar as APIs com base em um conjunto de métricas específicas de desempenho e escalabilidade. Além disso, o framework possibilita a análise e comparação resultados com o apoio estatístico, aumentando assim a credibilidade e confiabilidade dos resultados.

1.1 Escopo

Nesta dissertação, será realizada a concepção e o desenvolvimento de um framework, para avaliação das APIs do paradigma POO de manipulação de ontologias, levando em consideração dois aspectos principais: desempenho e escalabilidade. Além disso, o framework auxiliará os desenvolvedores no processo de análise e comparação dos resultados.

Neste sentido, não faz parte do escopo do trabalho avaliar as APIs referentes à manipulação de tripas RDF. Além disso, este trabalho não considera todas as implicações do processo de avaliação das APIs, como por exemplo, o isolamento da máquina que esta sendo executada, otimizações no código de implementação das aplicações, variações de hardware, aplicações sendo executadas em concomitância, consistência do modelo de dados utilizados, dentre outras. Tais fatores podem ser decisivos, pois possuem impacto no processo de avaliação das APIs e devem ficar a cargo do desenvolvedor. Isso é importante, pois diminui o risco de resultados tendenciosos na avaliação.

Embora o framework faça uso de métodos estatísticos para visualização e análise dos resultados, não faz parte do escopo deste trabalho definir, nem alterar os métodos já consolidados na literatura.

Comparar duas APIs pode ser uma tarefa difícil, pois as APIs utilizam estratégias de implementação e objetivos diferentes. Entretanto, o framework foi concebido sob a perspectiva dos desenvolvedores, sendo assim os mesmos podem decidir quando é plausível

[3] O acrônimo JOINT é devido ao projeto *Java Ontology Integration Toolkit* vinculado ao W3C, sendo o JOINT-E um módulo deste projeto.

comparar as APIs. Dessa maneira, o framework se posiciona como uma solução de apoio no processo de decisão do desenvolvedor.

Por fim, apesar do processo de tomada de decisão envolver outros fatores, tais como, flexibilidade e integração das APIs, suporte às consultas, documentação e clareza no mapeamento entre ontologia-objeto, estes não são considerados parte do escopo do trabalho.

1.2 Objetivos

O principal objetivo deste trabalho é a concepção e o desenvolvimento de um framework centrado na arquitetura que proporcione aos desenvolvedores facilidade na avaliação das APIs de manipulação de ontologias do paradigma POO. Este framework visa disponibilizar serviços de avaliação de desempenho e escalabilidade, flexibilização nas métricas de avaliação e comparação de dados. Desta forma, o trabalho lida tanto com a avaliação de uma única API quanto com a comparação com outras. Os objetivos específicos deste trabalho são:

- Reposicionar as métricas de avaliação de desempenho sob a perspectiva dos desenvolvedores que utilizam essas APIs do paradigma POO;
- Reposicionar as métricas de avaliação de escalabilidade sob a perspectiva dos desenvolvedores que utilizam essas APIs do paradigma POO;
- Desenvolver um framework para facilitar o processo de avaliação e comparação das APIs;
- Validar a proposta por meio de um experimento utilizando as duas das principais APIs (Alibaba e Jastor) utilizadas pelos desenvolvedores, fornecendo um guia útil para os mesmos.

1.3 Organização do texto

Esta dissertação esta dividida em cinco capítulos. O capítulo 1 apresenta o contexto, a motivação e os objetivos que compõem este projeto de mestrado, destacando os problemas atuais, bem como os resultados esperados a partir da condução do trabalho.

No Capítulo 2 são apresentados os conceitos relacionados ao tema deste trabalho. A teoria básica acerca de desenvolvimento de aplicações baseadas em ontologias, a metodologia de avaliação empregada, e uma breve explicação sobre alguns diagramas que serão utilizados para explicar o framework proposto, bem como detalhes da metodologia utilizada para programar o framework centrado na arquitetura. Além disso, também são apresentados os trabalhos relacionados.

O Capítulo 3 mostra como o JOINT-E foi criado descrevendo em detalhes cada um dos seus serviços e funcionalidades. Neste capítulo também é mostrado como o framework foi implementado, para isso são usados vários tipos de diagramas UML.

Depois no Capítulo 4, o experimento é apresentado, onde são detalhados alguns cenários para mostrar a flexibilidade do framework. Nele também é apresentado um questionário que foi aplicado com os desenvolvedores a fim de obter uma validação junto com os mesmos.

Por fim, no capítulo 5, são apresentadas as considerações finais deste trabalho, as limitações e os trabalhos futuros.

2 EMBASAMENTO TEÓRICO E TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar a fundamentação teórica referente ao foco deste trabalho, fazendo uma introdução sobre conceitos importantes da área que auxiliam na compreensão da pesquisa desenvolvida. Primeiramente, será apresentada uma conceituação sobre Web Semântica, destacando suas principais características. Subsequente, será apresentada uma introdução sobre ontologias e a seção seguinte apresenta paradigmas de desenvolvimento baseado em ontologias. Por sua vez, a seção 2.4 apresenta a abordagem utilizada por esse trabalho para definição dos objetivos de avaliação (desempenho e escalabilidade), chamada Goal-Question-Metric (GQM). A Seção 2.5 apresenta os conceitos relacionados a framework enquanto que a Seção 2.6 introduz o processo de desenvolvimento centrado na arquitetura. A Seção 2.7 apresenta os conceitos de componentes de software, bem como o modelo de componente adotado neste trabalho. Por fim, a Seção 2.8 explana sobre os trabalhos relacionados.

2.1 Web Semântica

A Web atual ainda é composta por uma grande quantidade de páginas estáticas ou dinamicamente geradas que se interligam. Tais páginas são feitas em uma linguagem chamada Hyper Text Markup Language (HTML), bastante utilizada para disponibilizar informações que são consumidas principalmente por humanos. As pessoas podem ler estas páginas e compreende-las, todavia, os dados contidos nestas páginas não possuem um significado associado de modo que possibilite a interpretação semântica dos mesmos por parte dos computadores. Neste cenário, surge a Web Semântica, que tem como ideia original estender a Web atual, descrevendo os dados e documentos atuais para que máquinas possam também entender e processar essa vasta coleção de informação (HEPP et al., 2008).

A Web Semântica foi concebida inicialmente por Tim Berners-Lee (BERNERS-LEE; HENDLER; LASSILA, 2001), onde a mesma nasceu com o propósito de representar informação de tal forma que ela possa ser utilizada por máquinas não apenas com o intuito de exibir os dados, mas para automatização, integração e reuso através de diferentes aplicações (BOLEY; TABET; WAGNER, 2001). Desta forma, a Web Semântica busca trazer significado para os conteúdos presentes nas páginas Web, criando um ambiente onde agentes de software percorre de página em página podendo facilmente realizar tarefas sofisticadas para os usuários finais, como agendamento de consultas médicas ou compras de pacotes de turismo. Tim Berners-Lee também propôs um modelo de camadas que foi e ainda é revisto pelo World Wide Web Consortium (W3C). O modelo pode ser visto na Figura 1.

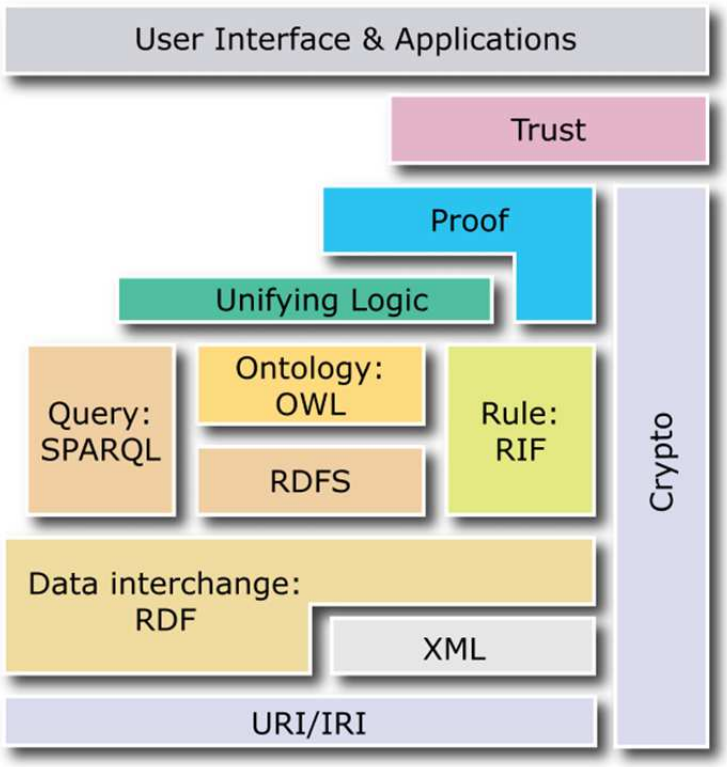

Figura 1 – As camadas da Web Semântica

Os principais componentes deste modelo, recomendados pela W3C, são descritos a seguir:

- RDF: O Resource Description Framework (RDF) é um framework para a criação de declarações em forma das conhecidas triplas RDF. Ele permite representar informações sobre os recursos na forma de grafos. A estrutura de qualquer elemento em RDF é vista como uma tripla da forma sujeito, predicado, objeto e estes elementos possuem um identificador associado, chamado URI, possibilitando assim ser descoberto por máquinas.

- RDFS: O Resource Description Framework Schema (RDFS) é uma extensão semântica ao RDF e é responsável por prover um conjunto de recursos interrelacionados. O objetivo do RDFS é proporcionar um conjunto bá sico de recursos que permita que documentos RDF sejam estruturados e validados. Utilizando RDFS é possível, por exemplo, criar hierarquias de classes e propriedades e descrever relacionamentos entre recursos.

- OWL: A Web Ontology Language (OWL) estende RDFS adicionando construcções mais avanc¸adas para descrever a sema^ntica de declarac¸o˜ es RDF. Ele permite expressividade de alto nível e inferência implícita e também possui restrições adicionais, como por exemplo, cardinalidade, restrições de dominio e imagem e regras

de união, disjunção, inversão e transitividade. Ele é baseado em lógica descritiva e por isso traz poder de raciocínio para a Web Semântica.

- SPARQL: O Simple Protocol and RDF Query Language (SPARQL) nada mais é do que uma linguagem de consulta para grafos baseado em RDF, ou seja, grafos na forma de triplas (Sujeito, Predicado, Objeto). SPARQL é utilizado também para extrair subgrafos de um RDF.

2.2 Ontologias

A Web Semântica se baseia em ontologias como suporte formal para representação de conhecimento e a Comunicação entre agentes de software. Desta forma, as ontologias possuem um papel fundamental para que a Web Semântica possua um plano de conhecimento explicitamente declarado. A palavra ontologia vem do Grego ontos e logos, conhecimento do ser (CASTRO, 2008). Em filosofia, ontologia refere-se ao estudo do ser enquanto ser.

Em Computação, de maneira informal, uma ontologia define um conjunto de conceitos e suas relações. Por exemplo, a terminologia (vocabulário do domínio), definição explícita dos conceitos essenciais, suas classificações, taxonomias, relações e axiomas dos domínios, incluindo hierarquias e restrições (DEVEDZIC, 2006). Formalmente, uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada (GRUBER, 1993). Para melhor entendimento, abaixo seguem detalhes sobre os termos citados nesta definição:

- Explícita: definições de conceitos, relações, restrições e axiomas;
- Formal: compreensível para agentes e sistemas;
- Conceitualização: modelo abstrato de uma área de conhecimento. Ressalta-se que a conceitualização está relacionado ao modelo estendido proposto por (GUARINO, 1998), onde uma conceitualização possui relações conceituais, ao invés de relações ordinárias;
- Compartilhada: conhecimento consensual.

Pode-se observar com as características supracitadas sobre ontologias que o conhecimento é explícito. Este conhecimento equivale à descrição de determinada área, garantindo um conhecimento "consensual" sobre a mesma. Várias ontologias são construídas de maneiras colaborativas, entre elas destacam-se: Friend of a Foaf (FOAF[1]), Semanticall Interlinked Online Communities (SIOC[2]), dentre outras. . A Figura 2.2 apresenta como as entidades da ontologia SIOC se relacionam.

[1] Mais informações em: <http://xmlns.com/foaf/spec/>
[2] Mais informações em: <http://sioc-project.org/ontology>

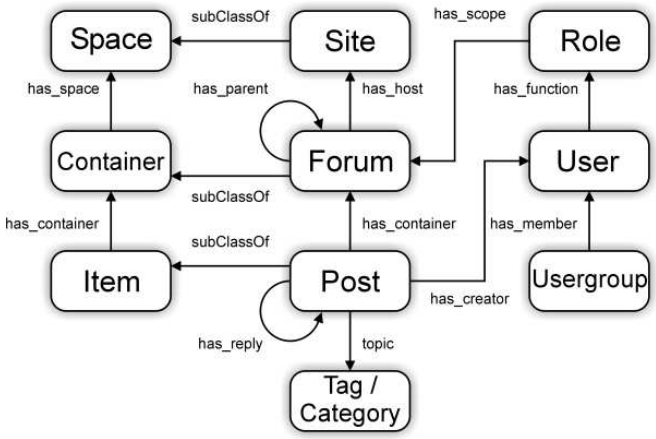

Figura 2 – Ontologia SIOC

A partir do momento que ha um consenso entre os engenheiros do conhecimento, ha a possibilidade de representar e compartilhar tal conhecimento através de uma ontologia e, consequentemente, integrá-la a outras áreas de conhecimento (através de outras ontologias), como mostrado na Figura 2.3.

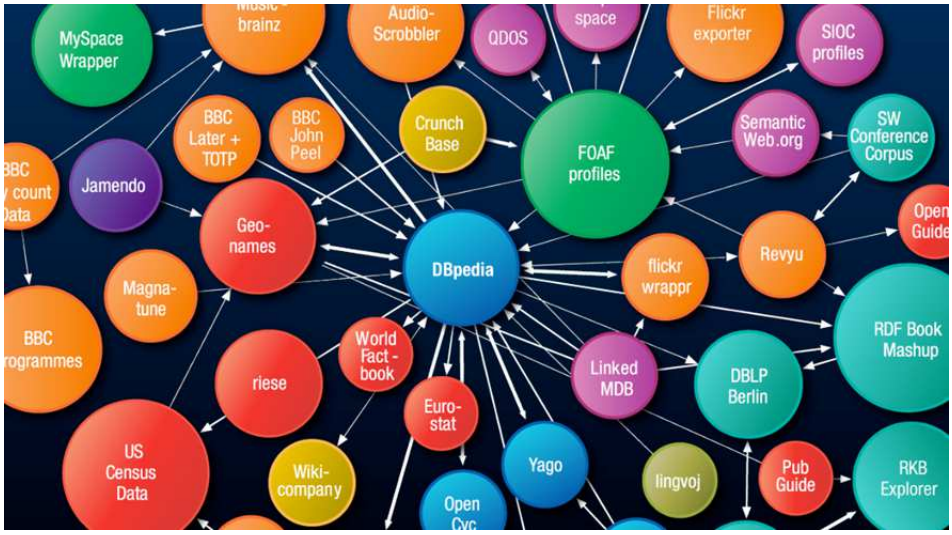

Figura 3 – Compartilhamento e integração de ontologias

Além disso, as ontologias tem o objetivo de serem expressas em uma linguagem formal que permite a compreensão por máquinas, provendo assim a automatização de atividades.

2.3 Desenvolvimento de Aplicações baseadas em Ontologias

A manipulação de instancias é um importante passo no processo de desenvolvimento de aplicações baseadas em ontologias. Atualmente existem duas principais abordagens utilizadas pelas ferramentas: desenvolvimento em triplas RDF e desenvolvimento orientado a objetos. Nas próximas seções, as principais distinções e benefícios das abordagens mencionadas serão apresentados.

2.3.1 Desenvolvimento orientado a triplas RDF

A maioria das APIs atuais ainda trabalha com o desenvolvimento baseado em triplas RDF. Dessa forma, os desenvolvedores da aplicação devem estar cientes de como funciona a ontologia em RDF, para poder manipular os dados através de cada tripla (sujeito, predicado e objeto) em código.

Caso um desenvolvedor deseje adicionar em uma ontologia, uma classe com várias propriedades inerentes a ela faz-se necessário escrever várias linhas de código representando cada tripla da classe, que captura apenas um valor para cada propriedade. Similarmente, se deseja remover esse recurso, várias triplas devem ser removidas.

Por exemplo, várias linhas de código são necessárias para criar a instância Alice, da entidade Person com a propriedade de name tendo valor "Alice", usando a API do Sesame, que manipula instâncias através do desenvolvimento baseado em triplas RDF. A Figura 2.4 ilustra como adicionar este recurso em um repositório do Sesame.

```
...
ValueFactory f = myRepository.getValueFactory();

// create some resources and literals to make statements out of
URI alice = f.createURI("http://example.org/people/alice");
URI name = f.createURI("http://example.org/ontology/name");
URI person = f.createURI("http://example.org/ontology/Person");
Literal alicesName = f.createLiteral("Alice");

RepositoryConnection con = myRepository.getConnection();
// alice is a person
con.add(alice, RDF.TYPE, person);
// alice's name is "Alice"
con.add(alice, name, alicesName);
...
```

Figura 4 – Exemplo de código da ferramenta Sesame

2.3.2 Desenvolvimento orientado a objetos

Diferentemente de triplas RDF, aplicações orientadas a objetos manipulam dados em nível de objetos e seus atributos. Tais objetos são caracterizados por um conjunto de atributos e valores. Neste sentido, faz-se necessária uma ferramenta que "traduza" as operações em objetos para a infraestrutura de triplas RDF da camada inferior. Algumas ferramentas foram criadas para prover esse paradigma para manipulação de ontologias. Com isso, os desenvolvedores não precisarão ter um conhecimento profundo da linguagem de representação da ontologia. Ou seja, uma classe no código representa uma classe na ontologia, sendo seus atributos mapeados como propriedades na ontologia.

Quando o desenvolvedor deseja adicionar na ontologia uma classe com várias propriedades, basta adicionar sua classe equivalente no código da aplicação, facilitando, assim, o desenvolvimento. Em comparação com o exemplo do Sesame (Figura 2.4), a Figura 2.5

mostra o mesmo recurso sendo adicionado ao repositório do Sesame usando a ferramenta Alibaba, que permite o desenvolvimento baseado no paradigma de orientação a objetos.

```
...
ObjectConnection con = repository.getConnection();

// create a Person
Person alice = new Person();
alice.setName("Alice");

// add a Person to the repository
con.addObject(alice);
...
```

Figura 5 – Exemplo de código da ferramenta Alibaba

2.4 Metodologia *Goal-Question-Metric*

A abordagem GQM (do inglês Goal-Question-Metric) é usada para encontrar e definir métricas para um determinado ambiente, em vez de simplesmente apresentar uma coleção de métricas mensuráveis. A abordagem GQM ajuda a identificar as razões pelas quais métricas particulares são escolhidas. A Figura 2.6 mostra a abordagem GQM (BASILI; CALDEIRA; ROMBACH, 1994).

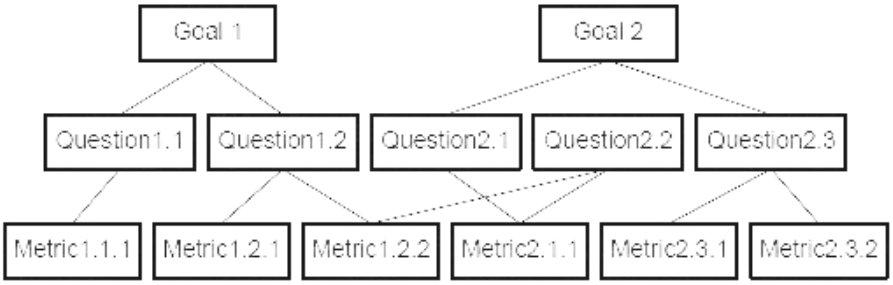

Figura 6 – Metodologia GQM (*Goal-Question-Metric*)

A abordagem GQM em uma perspectiva top-down envolve três passos:

- Passo 1 - Objetivo: é definido por meio de um modelo que consiste das seguintes partes

  Propósito - O que deve ser alcançado pela medição?

  Apsecto - Quais características devem ser medidas?

  Objeto - Qual artefato será avaliado?

  Perspectiva - A partir de qual perspectiva o objetivo vai ser definido?

  Contexto - Em que circunstâncias esta o objeto inserido?

- Passo 2 - Questão: quando respondidas, as questões proveem informações as quais ajudam a atingir o objetivo definido.

- Passo 3 - Métrica: para responder as questões, a abordagem GQM utiliza um conjunto de métricas que funcionam como valores quantitativos desta abordagem.

2.5 *Framework*

Existem várias definições de framework, sendo que as mais famosas são as (JOHNSON; FOOTE, 1988; JOHNSON, 1997), que dizem: i) um framework é um conjunto de classes que inclui um projeto abstrato para soluções de uma família de problemas relacionados; ii) Um framework é um conjunto de objetos que colaboram para realizar um conjunto de responsabilidades para um domínio de aplicação, um esqueleto de uma aplicação que pode ser customizada pelo desenvolvedor.

Destas definições podemos listar algumas características de um framework:

- Reusa o projeto de software e não apenas o código;

- Não é uma aplicação, e sim, um arcabouço para construir aplicações num domínio específico;

- É um framework voltado para um determinado domínio;

- Resolve problemas de uma mesma natureza;

- São aplicações incompletas.

Os frameworks podem ser classificados de várias maneiras, por exemplo, em relação a onde ele é usado (SAUVE, 2000):

- ***Framework* de suporte**: Prover serviços de nível de sistema operacional, como acesso aos arquivos ou computação distribuída;

- ***Framework* de aplicação**: São também conhecidos como frameworks horizontais. Esses fornecem funcionalidades que são aplicadas a uma variedade de problemas. São utilizados em mais de um domínio. Por exemplo, um framework para construção de interface GUI;

- ***Framework* de domínio**: São também conhecidos como frameworks verticais. Esses fornecem funcionalidades que são aplicadas num domínio específico. Por exemplo, framework para construir aplicações de controle de manufatura.

Eles também podem ser classificados em relação à técnica de extensão (FAYAD; SCHMIDT; JOHNSON, 1999):

- ***Framework* caixa branca**: Estão fortemente ligado as características das linguagens, para realizar a customização utilizam-se de herança. Requer um bom entendimento do framework para criar uma aplicação;

- ***Framework* caixa preta**: São instanciados a partir de algum tipo de configuração, como XML por exemplo. Esses frameworks confiam principalmente em composição (classes ou componentes) e delegação para realizar customização. Não requer entendimento de detalhes internos para produzir uma aplicação;

- ***Framework* caixa cinza**: São frameworks projetados para evitar as desvantagens apresentada por framework caixa branca e caixa preta, permitindo certo grau de flexibilidade e extensibilidade sem expor informações internas desnecessárias.

2.6 Desenvolvimento Centrado na Arquitetura

A arquitetura de software é um artefato essencial no ciclo de vida do software e deve auxiliar todas as suas fases (CHEN et al., 2002). A principal motivação para isso é que a abstração oferecida pela arquitetura de software possibilita uma análise estrutural em alto nível e um melhor entendimento do sistema.

Um processo de desenvolvimento centrado na arquitetura, além de usufruir dessa abstração arquitetural em todas as fases do desenvolvimento, deve fornecer meios de especificar a arquitetura do sistema, obedecendo as restrições impostas pelos seus requisitos não funcionais (YOU-SHENG; YU-YUN, 2003). Além disso, enquanto os processos tradicionais consideram a composição do sistema como sendo uma atividade de implementação (CHESSMAN; DANIELS, 2000), nos processos centrados na arquitetura, a composição dos componentes deve ser considerada em diferentes fases do desenvolvimento. Em outras palavras, a composição deve ser pensada nos diversos níveis de abstração (modelo abstrato, especificação detalhada e implementação) (CHEN et al., 2002).

As principais vantagens decorrentes da adoção de um processo de desenvolvimento centrado na arquitetura são: i) facilidade de se especificar propriedades não funcionais no sistema; ii) reutilização em um nível maior de granularidade; iii) redução do tempo de desenvolvimento; iv) representação estrutural explícita; v) compreensão facilitada do sistema; e vi) facilidade de manutenção.

Motivadas principalmente pela reutilização de software em larga escala, é cada vez maior o número de empresas que utilizam alguma técnica de desenvolvimento centrado na arquitetura (KOSKIMIES, 2001).

2.7 Desenvolvimento Baseado em Componentes

A grande utilização de Desenvolvimento Baseado em Componentes (DBC) está sendo motivada principalmente pelas pressões sofridas na indústria de software por prazos mais

curtos e produtos de maior qualidade. No DBC, uma aplicação é construída a partir da composição de componentes de software que já foram previamente especificados, construídos e testados, o que proporciona um ganho de produtividade e qualidade no software produzido.

Esse aumento da produtividade é decorrente da reutilização de componentes existentes na construção de novos sistemas. Já o aumento da qualidade é uma consequência do fato dos componentes utilizados já terem sido empregados e testados em outros contextos. Porém ale a pena ressaltar que apesar desses testes prévios serem benéficos, a reutilização de componentes não dispensa a execução dos testes no novo contexto onde o componente esta sendo reutilizado.

Apesar da popularização atual do DBC, não existe um consenso geral sobre a definição de um componente de software, que é a unidade básica de desenvolvimento do DBC. Porém, um aspecto muito importante é sempre ressaltado na literatura: um componente deve encapsular dentro de si seu projeto e implementação, além de oferecer interfaces bem definidas para o meio externo. O baixo acoplamento decorrente dessa política proporciona muitas flexibilidades, tais como: i) facilidade de montagem de um sistema a partir de componentes COTS[3]; e ii) facilidade de substituição de componentes que implementam interfaces equivalentes.

Uma definição de componentes, adotada na maioria dos trabalhos publicados atualmente, foi proposta em 1998 (SZYPERSKI, 1998). Segundo Szyperski, um componente de software é uma unidade de composição com interfaces especificadas através de contratos e dependências de contexto explícitas. Sendo assim, além de explicitar as interfaces com os serviços oferecidos (interfaces providas), um componente de software deve declarar explicitamente as dependências necessárias para o seu funcionamento, através de suas (interfaces requeridas).

Além dessa distinção clara entre interfaces providas e requeridas, os componentes seguem três princípios fundamentais, que são comuns à tecnologia de objetos (CHESSMAN; DANIELS, 2000):

1. **Unificação de dados e funções:** Um componente deve encapsular o seu estado (dados relevantes) e a implementação das suas operações oferecidas, que acessam esses dados. Essa ligação estreita entre os dados e as operações ajuda a aumentar a coesão do sistema;

2. **Encapsulamento:** Os clientes que utilizam um componente devem depender somente da sua especificação, nunca da sua implementação. Essa separação de interesse [4] reduz o acoplamento entre os módulos do sistema, além de melhorar a sua manutenção;

[3] do inglês *Common-Off-The-Shelf*
[4] do inglês *separation of concerns*

3. **Identidade:** Independentemente dos valores dos seus atributos de estado, cada componente possui um identificador único que o difere dos demais.

Existem várias tecnologias para se implementar componentes, entre as mais famosas estão: CORBA do OMG (Object Management Group), COM/COM+ da Microsoft e Entreprise JavaBeans (EJB) da Sun. Contudo, todas elas seguem um modelo dependente de plataforma. Por essa razão decidimos utilizar o COSMOS* no trabalho apresentado. A próxima seção vai apresentar as motivações da metodologia COSMOS*, suas vantagens e implementação.

2.7.1 Metodologia COSMOS*

O modelo COSMOS* (GAYARD; RUBIRA; GUERRA, 2008) usa recursos de linguagem de programação e padrões de projeto amplamente conhecidos para representar arquiteturas de software explicitamente no código do programa. Este modelo define a especificação e implementação de componentes, conectores, configurações arquiteto^ nicas e componentes compostos.

Ele é dividido em um modelo de especificação, pelo qual um componente expõe as suas especificações; um modelo de implementação, que orienta a implementação interna de um componente, um modelo de conector, que especifica a conexão entre os componentes através de conectores, e um modelo de composição, pelo qual novos componentes ou sistemas inteiros são construídos utilizando componentes existentes.

Portanto, os principais objetivos do modelo COSMOS* são: fornecer um acompanhamento a partir da arquitetura de software até código do programa e facilitar na manutenção do software. Este modelo reduz o acoplamento entre os módulos da arquitetura e ainda oferece reutilização de software. Embora COSMOS* tenha quatro submodelos (especificação, implementação conector, e composição), este trabalho utiliza apenas dois que são o modelo de especificação e modelo de implementação, pois os conectores são classes simples. É importante citar todos os componentes do framework foram implementados usando COSMOS*.

A figura mostra um exemplo de um componente especificado com o COSMOS*. Nesta figura podemos identificar as principais propriedades de um componente especificado com o COSMOS*. Elas são:

- A interface provida (IA na figura) é a única parte do componente que o usuário tem acesso direto. Dessa forma para utilizar o componente ele não precisa saber de detalhes da implementação;

- A interface requerida (IB na figura) representa a restrição do componente. Para ele funcionar adequadamente o usuário tem que passar uma classe que implemente essa interface;

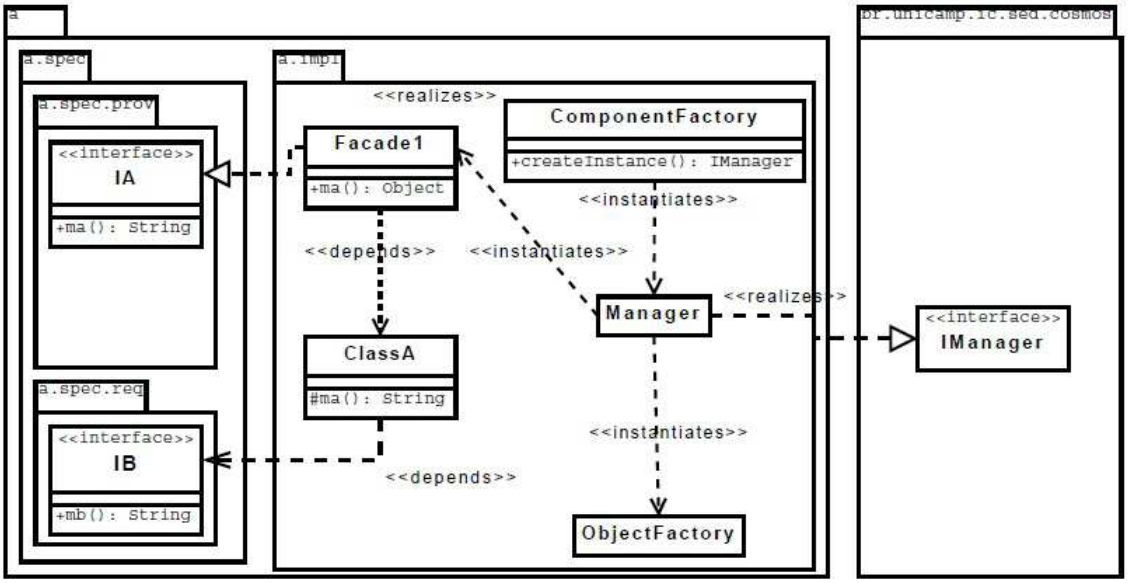

Figura 7 – Exemplo de Componente COSMOS*

- A classe *Facade1* implementa a interface provida e funciona como o padrão de projeto facade. Em outras palavras, essa classe é responsável por delegar as funções da interface provida para as classes que realmente implementam elas;
- A classe *componenteFactory* implementa o padrão de projeto *Factory Method*. Em outras palavras, é responsável por instanciar as classes internas do componente;
- A *classA* neste exemplo é a classe que implementa os serviços do componente. Como mostra a figura ela depende diretamente da interface requerida;
- Por fim, temos a classe manager, que é responsável por gerenciar todo acesso ao componente.

Todos os serviços criados no framework proposto nesse trabalho seguem o modelo COSMOS*. Em outras palavras, eles são componentes e a implementação deles é muito parecida com a figura 7.

2.8 Trabalhos Relacionados

Na literatura existem alguns trabalhos que focam em avaliações isoladas de algumas APIs. Entretanto, há poucos com foco nas APIs do paradigma POO e menos ainda trabalhos focados nos desenvolvedores.

Em (MAGKANARAKI et al., 2002) apresenta uma análise estatística sobre o tamanho e a morfologia das ontologias em RDFS, enquanto que (ALEXAKI et al., 2001) desenvolveu um benchmark utilizando consultas em estruturas RDF. Por sua vez, (GUO;

PAN; HEFLIN, 2005) também apresenta um benchmark focado em consultas em ontologias. Dessa maneira, esses trabalhos focam apenas no acesso, por outro lado, o JOINT-E considera outras questões além, como, inserção, atualização e remoção.

(THEOHARIS; CHRISTOPHIDES; KARVOUNARAKIS, 2005) mostra um benchmark com três diferentes bases de RDFS, sendo elas: i) representação em tabelas, onde uma tabela era uma classe ou propriedade; ii) uma única tabela representando todas triplas (sujeito-predicado-objeto); um sistema de híbrido, onde os autores fizeram uma composição entre o modo i e ii. Embora os autores mostrem como diferentes representações alteram a desempenho das aplicações, os mesmos não focam em representações no nível de objeto.

Por um lado, (ABADI et al., 2007) avalia o requisito escalabilidade, examinando as soluções de manipulação de ontologias em RDF, explorando as limitações de cada solução. Nesse sentido, os autores utilizam consultas a fim de avaliar a escalabilidade de cada tecnologia.

Por outro lado, (HERTEL; BROEKSTRA; STUCKENSCHMIDT, 2009) avalia escalabilidade e desempenho das tecnologias utilizando consultas. Enquanto que, o trabalho proposto examina escalabilidade usando cenários de execução, onde são monitorados os recursos (memória e CPU) do servidor, a fim de avaliar a escalabilidade.

Embora os trabalhos relacionados avaliam os aspectos de escalabilidade e desempenho, nenhum avalia esses requisitos no nível de objeto ou cobre todos os fatores apresentados neste trabalho.

3 JOINT-E

Durante o processo de desenvolvimento de aplicações semânticas, deve-se selecionar a API mais adequada de acordo com as necessidades e requisitos de cada aplicação. Assim, os desenvolvedores precisam avaliar essas APIs com métricas e métodos adequados para fundamentar suas decisões. Neste contexto, o framework proposto, chamado JOINT-E, foi desenvolvido com base no método de avaliação elaborado por (GARCIA-CASTRO; GOMEZ-PEREZ, 2005). Além disso, é importante salientar que o framework proposto focou-se nos seguintes aspectos:

- Avaliar a camada de persistências (isto é, I/O na ontologia), em vez de avaliar toda a aplicação;
- Focar na avaliação de APIs de persistências de alto nível que fornecem um mapeamento entre ontologia e objetos;
- Seguir as diretrizes propostas por (GUO; PAN; HEFLIN, 2005) para avaliar o *workload*;
- Adotar um benchmark para avaliar as APIs de persistência, considerando aspectos, como: inserção, recuperação, atualização e exclusão nas ontologias;
- Definir métricas de desempenho baseadas em (MENASCE; ALMEIDA; DOWDY, 2004), considerando tanto o tempo de execução quanto o consumo de recursos de máquina (ou seja, CPU e memória). Frisa-se que essas métricas foram reposicionadas para o escopo das APIs de persistência em ontologia do paradigma POO;
- Definir métricas de escalabilidade baseadas em (ZIMMERMANN et al., 2005). Assim, os cenários que envolvem um número crescente de requisições foram definidos, a fim de avaliar se a arquitetura das APIs pode ser configurada para suportar uma quantidade crescente de requisições. É importante ressaltar que esses cenários foram adaptados ao escopo das APIs de persistência em ontologia do paradigma POO.

3.1 Planos GQM

Com o intúito de fundamentar e definir as métricas/critérios de avalição que farão parte do JOINT-E, este trabalho faz uso da metodologia GQM (Goal-Question-Metric) (BASILI; CALDEIRA; ROMBACH, 1994) para oferecer uma descrição clara das métricas de desempenho e escalabilidade que lidam com as características inerentes das APIs com paradigma POO. Esta metodologia foi utilizada na definição dos objetivos de avaliação,

apresentando como cada métrica auxilia a responder um objetivo desejado. Nesta Seção, primeiro será apresentado o plano GQM referente aos critérios de desempenho e, em seguida, aos critérios de escalabilidade. Ressalta-se que os planos GQM foram elaborados para avaliar as operações criar, recuperar, atualizar e apagar (CRUD) em diferentes cenários.

3.1.1 Plano GQM - Desempenho

O desempenho é uma das características mais importantes das APIs de persistência, sendo um requisito essencial nas aplicações, influenciando diretamente na interação humano-computador. O plano GQM para o desempenho foi baseado em (MENASCE; ALMEIDA; DOWDY, 2004), onde é analisado o desempenho relacionado a três aspectos diferentes: tempo de resposta, utilização de recursos e produtividade.

$Objetivo_1$
Propósito: Avaliação
Objeto: Diferentes APIs de persistência em ontologias
Aspecto: Desempenho
Perspectiva: Time de desenvolvimento de software
Contexto: APIs de persistência em ontologias do paradigma OO

Para o plano de GQM de desempenho, as seguintes questões foram elaboradas a fim de cobrir este objetivo.

$Questão_{1.1}$: Qual é a capacidade de resposta de uma API POO, em cada a operação CRUD, obedecendo uma determinada configuração estabelecida pelo desenvolvedor?

Para responder a questão 1.1, as métricas seguintes foram elaboradas:

Métricas
$M_{1.1.1}$: Tempo da operação *criar uma instância*
$M_{1.1.2}$: Tempo da operação *recuperar uma instância*
$M_{1.1.3}$: Tempo da operação *atualizar uma instância*
$M_{1.1.4}$: Tempo da operação *apagar uma instância*

As métricas acima são mensuradas em milissegundo e representam o tempo total requerido para ser executada uma determina operação.

$Questão_{1.2}$: Qual é o comportamento da API POO, com relação a utilização de recursos de máquina, levando em consideração as operações CRUD, obedecendo uma determinada configuração estabelecida pelo desenvolvedor?

No processo de avaliação de desempenho das APIs de persistência POO, esta questão possui um papel importante, pois permite aos desenvolvedores entenderem melhor o comportamento das APIs avaliadas, com relação à utilização de recursos de máquina.

Métricas
$M_{1.2.1}$: Percentual de utilização de CPU no instante que a operação *criar uma instância* foi executada
$M_{1.2.2}$: Percentual de utilização de CPU no instante que a operação *recuperar uma instância* foi executada
$M_{1.2.3}$: Percentual de utilização de CPU no instante que a operação *atualizar uma instância* foi executada
$M_{1.2.4}$: Percentual de utilização de CPU no instante que a operação *apagar uma instância* foi executada
$M_{1.2.1}$: Quantidade de memória consumida durante a execução da operação *criar instância*
$M_{1.2.2}$: Quantidade de memória consumida durante a execução da operação *recuperar instância*
$M_{1.2.3}$: Quantidade de memória consumida durante a execução da operação *atualizar instância*
$M_{1.2.4}$: Quantidade de memória consumida durante a execução da operação *apagar instância*

A utilização de recursos pode ser representada de duas formas diferentes: utilização da CPU medida em porcentagem e utilização da memória medida em kilobytes.

Questão$_{1.3}$: Qual é a taxa de sucesso das operações CRUD de uma API em determinado espaço de tempo, obedecendo uma determinada configuração estabelecida pelo o desenvolvedor?

Por fim, a questão que compõe o plano GQM para desempenho está relacionada à produtividade das APIs. O objetivo nesta questão é monitorar a produtividade através da vazão (*throughput*) das APIs, ou seja, em determinado espaço de tempo é medida a quantidade de operações que foram realizadas com sucesso. Por exemplo, em um minuto verifica-se qual foi a quantidade de instâncias criadas corretamente.

Métricas
$M_{1.3.1}$: Vazão da operação *criar uma instância*
$M_{1.3.2}$: Vazão da operação *recuperar uma instância*
$M_{1.3.3}$: Vazão da operação *atualizar uma instância*
$M_{1.3.4}$: Vazão da operação *apagar uma instância*

3.1.2 Plano GQM - Escalabilidade

De acordo com (GARCIA-CASTRO; GOMEZ-PEREZ, 2005), a escalabilidade das APIs de persistência em ontologias é uma necessidade primária e precisa ser investigada cuidadosamente. Embora escalabilidade esteja, de certa forma, diretamente relacionada à desempenho é necessário um plano GQM diferente.

Diferentemente do plano GQM de desempenho, o plano GQM para escalabilidade é baseado em cenários, onde foram elaborados cenários cliente-servidor a fim de avaliar a escalabilidade. Além disso, as questões, bem como suas métricas foram desenvolvidas baseadas no trabalho de (ZIMMERMANN et al., 2005).

Objetivo$_2$
Propósito: Avaliação
Objeto: Diferentes APIs de persistência em ontologias
Aspecto: Escalabilidade
Perspectiva: Time de desenvolvimento de software
Contexto: APIs de persistência em ontologias do paradigma OO

O cenário elaborado é em relação ao número máximo de requisições suportadas pela API e está sendo representado pela questão 2.1.

Questão$_{2.1}$: Qual é o maior número de requisições simultâneas suportadas pela API?

Esta questão é comum quando se tenta medir a escalabilidade de um sistema construído usando uma API. É importante notar que as aplicações baseadas em ontologias precisam prover escalabilidade sem reduzir a qualidade dos serviços oferecidos (GARCIA-CASTRO; GOMEZ-PEREZ, 2005). Dessa maneira, para responder a esta questão, as seguintes métricas foram elaboradas:

Métricas
$M_{2.1.1}$: Número máximo de requisições
$M_{2.1.2}$: Tempo até o serviço atingir o máximo de requisições e ficar inoperante (i.e. até o serviço "cair")

A métrica $M_{2.1.1}$ representa o número de requisições respondidas com sucesso. Enquanto que a métrica $M_{2.1.2}$ mede o tempo até a API POO ficar indisponível.

A fim de aumentar a cobertura na avaliação da escalabilidade das APIs, a questão 2.2 foi elaborada.

Question$_{2.2}$: Qual API POO usa menos recurso de máquina (CPU e memóra) em um cenário considerado crítico do ponto de vista de escalabilidade?

Este cenário crítico tem por objetivo simular vários usuários simultâneos requisitando diferentes tipos de serviços. Dessa maneira, esta questão aborda como as APIs se comportam em situações críticas.

Métricas
$M_{2.2.1}$: Média de consumo de memória
$M_{2.2.2}$: O valor máximo de consumo da memória
$M_{2.2.3}$: Média do uso de CPU
$M_{2.2.4}$: O valor máximo de uso da CPU

As métricas elaboradas ajudam a entender o comportamento das API onde são medidos os valores médios e os picos no consumo. Vale ressaltar que os dois aspectos considerados são: CPU e memória.

3.2 Análise de dados

Como dito anteriormente, o JOINT-E é um framework que permite aos desenvolvedores avaliarem APIs que apoioam o desenvolvimento de aplicações semânticas de maneira rápida e simples. Além disso, framework também prove funcionalidades de análise e comparação dos dados com o apoio estatístico, aumentando assim a credibilidade dos resultados. O apoio estatístico vem através de testes de hipóteses, testes de normalização empregados nos dados obtidos e gráficos gerados.

Utilizando o JOINT-E, os desenvolvedores podem selecionar quais planos GQM serão executados. Além disso, poderão selecionar quais questões dentro de cada plano GQM serão respondidas, permitindo assim que os desenvolvedores tenham a flexibidade de analisar as APIs utilizando diferentes aspectos. Após o desenvolvedor selecionar os planos, bem como suas questões, cada métrica presente nas questões será executada trinta vezes. Este número é uma convenção para o tamanho da amostra proposto por (TRIOLA, 1999). Assim, após esse conjunto de execuções, cada métrica irá gerar uma distribuição de dados que caracteriza o resultado dessa execução. De posse dessas distribuições, o JOINT-E gera gráficos, previamente escolhidos pelos desenvolvedores, a fim de apresentar os resultados. Mais detalhes sobre o processo de execução do JOINT-E serão apresentados nas seções subsequentes.

Embora avaliar uma API isoladamente ajude os desenvolvedores a compreender melhor o comportamento desta API, em muitos casos é necessário comparar duas ou mais APIs em diferentes aspectos. Visando atender a essa necessidade, o JOINT-E provê um módulo onde os desenvolvedores poderão fazer uma comparação dois a dois relacionando cada aspecto avaliado. Ou seja, dentro de um universo de APIs disponíveis, os desenvolvedores escolherão quais questões (tempo de resposta, utilização de recursos, dentre outras) serão respondidas para realizar uma comparação. Dessa forma, o framework identifica as

APIs mais adequadas de acordo com as métricas de desempenho e escalabilidade sob a perspectiva dos desenvolvedores.

Para validar as respostas obtidas pelo framework, para cada questão de cada plano GQM, foi elaborado um conjunto de hipóteses. Dependendo do tipo de distribuição obtida, ou seja, uma distribuição normal ou não, será executado um teste de validação de hipótese. Dessa maneira, o teste de hipótese executado pode ser um teste paramétrico ou não paramétrico conforme discutiremos na proxima subseção.

3.2.1 Hipóteses para o Desempenho

Para validar os resultados relacionados as métricas de desempenho, é proposto três questões a serem respondidas, cujo mapeamento com hipóteses é apresentado abaixo.

Para $Questão_{1.1}$:

- $H_{1.0}$ Não existe diferença entre APIs POO, levando em consideração o aspecto *tempo de resposta*.
- $H_{1.1}$ Existe diferença entre APIs, levando em consideração o aspecto *tempo de resposta*.

Para $Questão_{1.2}$:

- $H_{2.0}$ Não existe diferença entre APIs POO, levando em consideração o aspecto *utilização de recurso*.
- $H_{2.1}$ Existe diferença entre APIs, levando em consideração o aspecto *utilização de recurso*.

Para $Questão_{1.3}$:

- $H_{3.0}$ Não existe diferença entre APIs POO, levando em consideração o aspecto *produtividade*.
- $H_{3.1}$ Existe diferença entre APIs, levando em consideração o aspecto *produtividade*.

3.2.2 Hipóteses para o Escalabilidade

Por sua vez, para escalabilidade, tem-se duas questões a serem respondidas, cujo mapeamento com hipóteses é apresentado abaixo:

Para $Questão_{2.1}$:

- $H_{1.0}$ Não existe diferença entre as APIs POO levando em consideração o aspecto o número de requisições respondidas com sucesso.
- $H_{1.1}$ Existe diferença entre as APIs POO levando em consideração o aspecto o número de requisições respondidas com sucesso.

Para $Questão_{2.2}$:

- $H_{1.0}$ Não existe diferença entre as APIs POO levando em consideração o aspecto o número de requisições ao comportamento das APIs em um cenário considerado crítico.
- $H_{1.1}$ Existe diferença entre as APIs POO levando em consideração o aspecto o número de requisições ao comportamento das APIs em um cenário considerado crítico.

3.3 Visão Geral

Por meio da metologia GQM e o uso de testes estatísticos para avaliar os resultados o JOINT-E oferece recursos interessantes para avaliar e comparar APIs. A Figura 8 apresenta os principais módulos do framework, bem como a interação entre os mesmos. Os módulos são: GUI (*Graphical User Interface*), serviços gerais, serviços de aplicação, ferramentas e persistência. Vale ressaltar que a interação dos módulos foi adaptada de (MELLO, 2011).

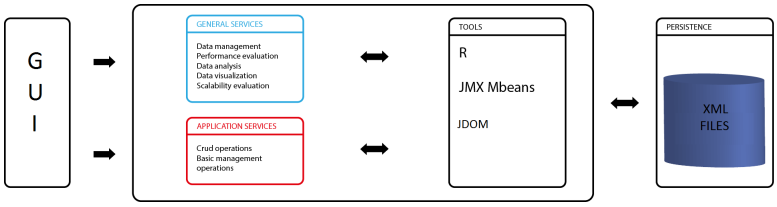

Figura 8 – Visão Geral do JOINT-E

- **GUI**: este módulo é o correspondente à interface gráfica responsável pela interação entre o desenvolvedor e o framework.
- **Módulo de Serviços Gerais**: este módulo implementa serviços básicos que são fornecidos no JOINT-E, tais como: análise estatística, implementação dos planos GQM, entre outros. É importante frisar que estes serviços equivalem aos *frozen spots* do framework.
- **Serviços de Aplicação**: principais serviços que são usados no JOINT-E, tais como: operações CRUD e operações de gerenciamento básico (recuperar total de instâncias, dentre outras). Os *hot spots* do framework estão nesse módulo;
- **Ferramentas**: este módulo contém um conjunto de ferramentas que são usadas para realizar os serviços dos outros módulos, tais como R (R Development Core Team, 2010), JMX Mbeans[1] e o JDOM (COMMUNITY, 2014).

[1] Mais detalhes em: <http://docs.oracle.com/javase/tutorial/jmx/mbeans/>. Último acesso: Janeiro, 2014

- **Persistência**: para cada execução das questões presentes nos planos GQMs, será gerado um arquivo XML respectivo, tal arquivo poderá ser utilizado pelo o desenvolvedor para gerar gráficos novamente, comparar os resultados provenientes de outras APIs, dentre outras funcionalidades. Através desse módulo os arquivos XML são gerados.

3.4 Projeto Arquitetural

Antes de explicar a arquitetura do *framework* é importante destacar algumas decisões de projeto que foram tomadas e suas consequências.

- ***Framework***: foi escolhido criar um framework porque o problema tratado neste trabalho possui pontos de variabilidade. Além disso, framework já é uma abordagem consolidada no que se refere ao reuso de software (JOHNSON, 1997);

- ***Framework* caixa cinza**: a vantagem de desenvolver um framework deste tipo é que o usuário não precisa conhecer todos os detalhes do código para instanciar o framework (como no caso de caixa branca) e ao mesmo tempo não fica "preso" ao que já está definido (como no caso de caixa preta) (FAYAD; SCHMIDT; JOHNSON, 1999);

- **Desenvolvimento centrado na arquitetura**: utilizar esta abordagem para criar o framework deixa cada componente bem modularizado, facilitando, por exemplo, a instanciação e evolução do sistema. Além disto, é bem mais simples de entender o código com os artefatos da arquitetura (YOU-SHENG; YU-YUN, 2003);

- **Desenvolvimento Baseado em Componentes(DBC)**: utilizar DBC traz duas vantagens principais: i) aumento da produtividade, decorrente da reutilização de componentes existentes na construção de novos sistemas; ii) aumento da qualidade, em consequência do fato dos componentes utilizados já terem sido empregados e testados em outros contextos (SZYPERSKI, 1998);

- **COSMOS***: a principal vantagem de se utilizar o COSMOS* no contexto desse trabalho é que todos os componentes desenvolvidos seguem esse modelo. Desta forma, qualquer desenvolvedor que esteja familiarizado com o COSMOS* pode facilmente entender o código do JOINT-E. No COSMOS* os conceitos de interfaces requeridas e conectores são explicitamente representados (GAYARD; RUBIRA; GUERRA, 2008).

A seguir será apresentado o modelo de features, explicitando os pontos de variação do framework. Depois será mostrado o diagrama de casos, apresentando as funcionalidades, sob a perspectiva do usuário. Por fim, será apresentado o diagrama de componentes do JOINT-E.

3.4.1 Modelo de Features

A Figura 3.2 apresenta o diagrama de features (KANG et al., 1990) do JOINT-E. O objetivo desse diagrama é trazer ao desenvolvedor um artefato visual dos pontos de variabilidade do JOINT-E. E Alguns pontos nesse diagrama devem ser destacados:

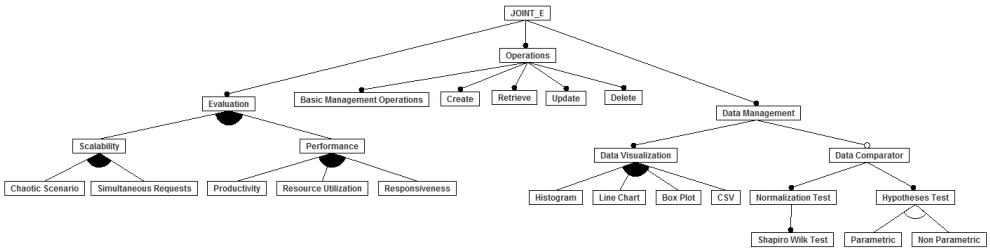

Figura 9 – Diagrama de *Features*

- As features *Evaluation, Operations* e *Data Management* são obrigatórias;
- Todas as features subtipos de *Operations* são obrigatórias.
- As features subtipos de *Evaluation* obedecem a lógica do operador **OR** não exclusivo, ou seja, os desenvolvedores podem escolher um ou mais atributos. Sendo assim, é possível avaliar *Performance* e *Scalability* separadamente ou juntos, em uma mesma instância do framework. O mesmo acontece com os subtipos de *Performance*.
- Embora a feature *Data Management* seja obrigatória, seus subtipos necessariamente não são. Como pode ser visto, a feature *Data Comparator* é opcional. Dessa maneira, os desenvolvedores podem utilizar o JOINT-E apenas para a avaliação de uma única API, não podendo assim comparar os resultados com outras APIs. Vale ressaltar que a feature *Data Visualization* possui a mesma lógica de variabilidade encontrada na feature *Evaluation*, onde possível selecionar um ou mais tipos de visualização de dados. Além disso, *Data Visualization* é uma feature obrigatória.

3.4.2 Casos de Uso

O JOINT-E é um framework que possibilita aos desenvolvedores diversas maneiras de avaliação, visualização de dados, dentre outras. A fim de explanar melhor as funcionalidades, a Figura 3.3 apresenta o diagrama de casos de uso.

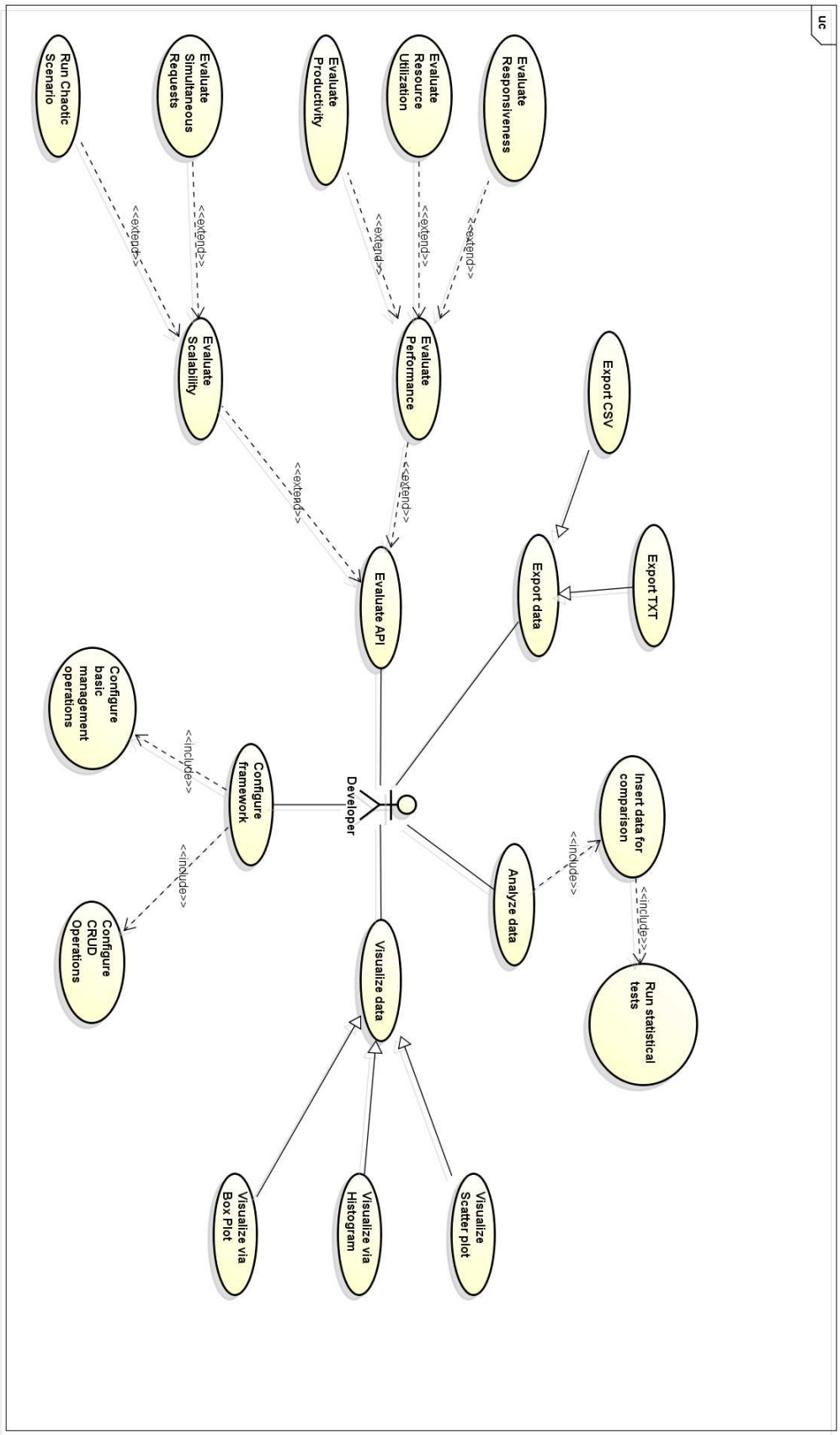

Figura 10 – Diagrama de Casos de Uso

A seguir será apresentada a narrativa dos principais casos de uso.

Caso de Uso - *Configure Framework*
Resumo: o desenvolvedor configura o framework.
Ator: o desenvolvedor.
Dependências: não possui.
Narrativa: este caso de uso é o primeiro passo para a execução correta da instância do framework. A configuração do framework compreende dois passos fundamentais: i) configurar as operações CRUD; e ii) configurar as operações de gerenciamento básico. O desenvolvedor deve implementar essas operações. Além disso, a configuração do framework compreende aspectos cadastrais, tais como: nome do desenvolvedor, configuração da máquina, dentre outras. Dessa maneira, esse caso de uso também compreende seleção de outras features, como por exemplo, formato dos dados exportados, tipos de visualização de dados, dentre outras. Mais detalhes sobre essas operações serão apresentados na Seção 3.7.

Caso de Uso - *Evaluate API*
Resumo: o desenvolvedor avalia uma API de persistência do paradigma OO.
Ator: o desenvolvedor.
Dependências: ter configurado o framework adequadamente para avaliar essa API.
Narrativa: este caso de uso começa quando o desenvolvedor quer avaliar uma API, previamente configurada. Nele, o desenvolvedor pode avaliar uma API, sob o aspecto Desempenho, onde o mesmo pode selecionar qual característica avaliar (capacidade de resposta, utilização de recurso e produtividade). Além disso, nesse caso de uso, o desenvolvedor pode avaliar a API sob o aspecto de Escalabilidade.

Caso de Uso - *Export data*
Resumo: o desenvolvedor exporta os dados obtidos nas avaliações das APIs.
Ator: o desenvolvedor.
Dependências: ter selecionado quais formatos serão exportados os dados. Vale ressaltar que esse passo é feito na configuração do framework.
Narrativa: este caso de uso começa quando o desenvolvedor precisar exportar os dados, para serem trabalhados em outra ferramenta. Nele o desenvolvedor tem a possibilidade de exportar os dados no formato CSV e TXT.

Caso de Uso - *Visualize data*
Resumo: o desenvolvedor visualiza os dados obtidos nas avaliações das APIs. **Ator:** o desenvolvedor. **Dependências:** ter selecionado quais os tipos de visualizações que serão geradas. Vale ressaltar que esse passo é feito na configuração do framework. **Narrativa:** este caso de uso compreende os vários tipos de visualização fornecidos pelo framework. Nele, o desenvolvedor pode visualizar os dados através de box-plot, histograma e gráfico de distribuição de valores.

Caso de Uso - *Analyze data*
Resumo: o desenvolvedor analisa os dados, comparando os dados obtidos nas avaliações das APIs. **Ator:** o desenvolvedor. **Dependências:** ter selecionado a *feature Comparador de dados.* **Narrativa:** este caso de uso começa quando o desenvolvedor quer comparar os dados das execuções de uma API com outra. Nele, o desenvolvedor pode inserir os dados obtidos nas execuções e em seguida executar as comparações, onde através de testes de hipóteses será mostrado qual API teve a melhor performance nos aspectos selecionados para a comparação.

3.4.3 Diagrama de Componentes

A Figura 3.4 mostra a arquitetura de componentes do framework. Ela segue um estilo de arquitetura baseado em componentes e camadas. As quatro camadas de alto nível foram definidas como: (i) Aplicação, contém a interface de interação com o framework, (ii) Sistema, contém os principais componentes do framework (kernel do sistema), (iii) Negócio, controla a persistência do framework, e (iv) Infra, representa a conexão com o repositório de ontologias.

A arquitetura do JOINT-E implementa todos os conceitos apresentados nas seções anteriores. É importante frisar que cada plano GQM foi encapsulado em um componente de controle (Desempenho e Escalabilidade), e as questões GQM divididas uma em cada componente. Dessa maneira, aumenta-se o reuso e diminui a granularidade dos componentes. Além disso, a variabilidade do framework fica melhor explicitada. Da mesma maneira, acontece com as features relacionadas a analise e a visualização de dados. Cada formato de visualização, bem com os testes estatísticos, está encapsulado em componentes diferentes.

Frisa-se que o componente Operações (em vermelho na figura) concentra os *hot spots* do framework. A implementação desse componente está em caráter parcial (mais detalhes sobre a implementação desse componente será apresentado na Seção 3.7).

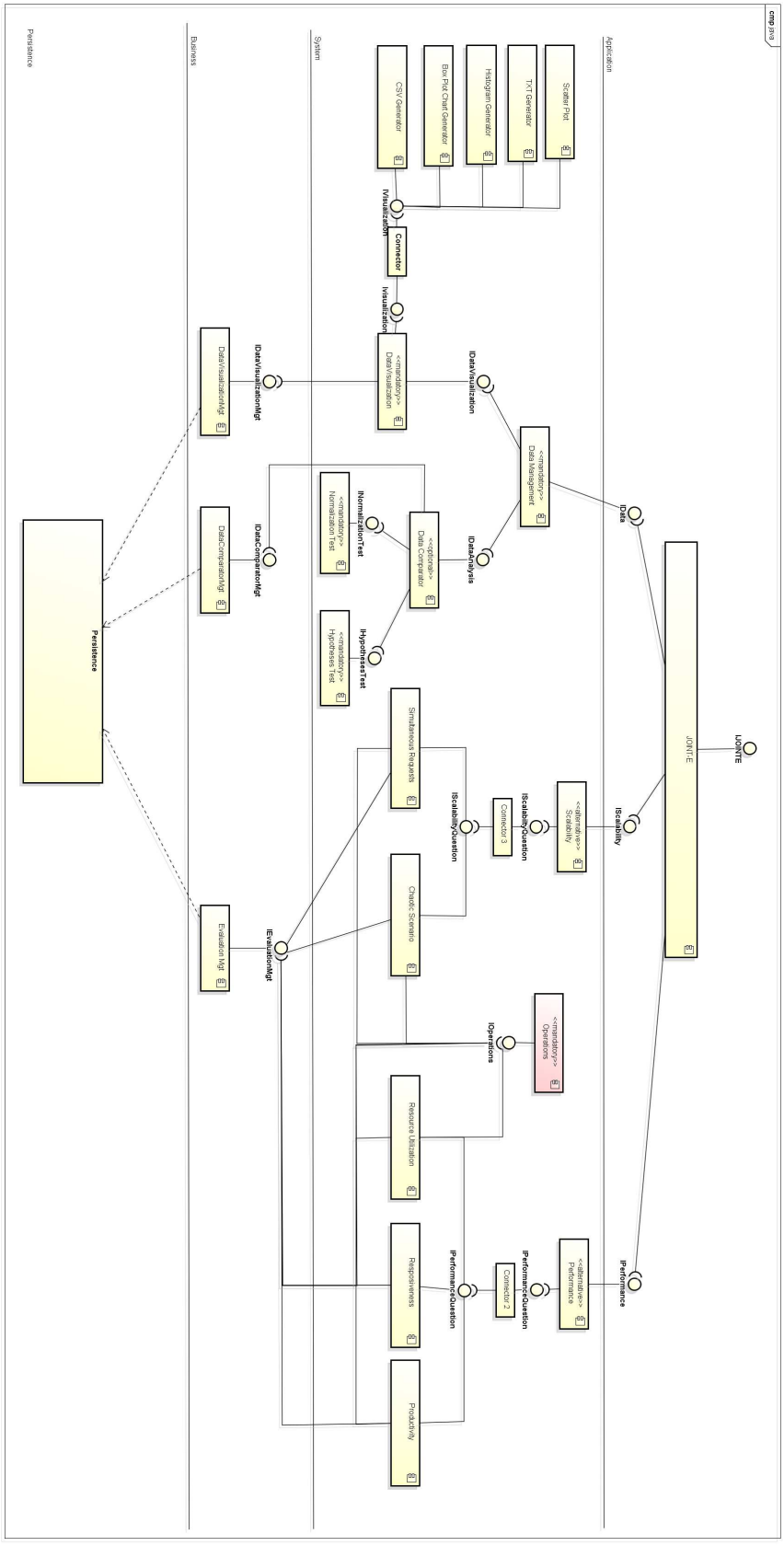

Figura 11 – Diagrama de Componentes

3.5 GUI

Este módulo corresponde à interface gráfica (GUI) do JOINT-E. Através dela, o desenvolvedor pode interagir com os serviços disponibilizados. A Figura 3.5 apresenta a tela principal da GUI. Pode-se observar que o desenvolvedor pode selecionar quais questões GQM, o mesmo deseja avaliar, bem como outros serviços, por exemplo, geração de gráfico, comparação de resultados, dentre outros.

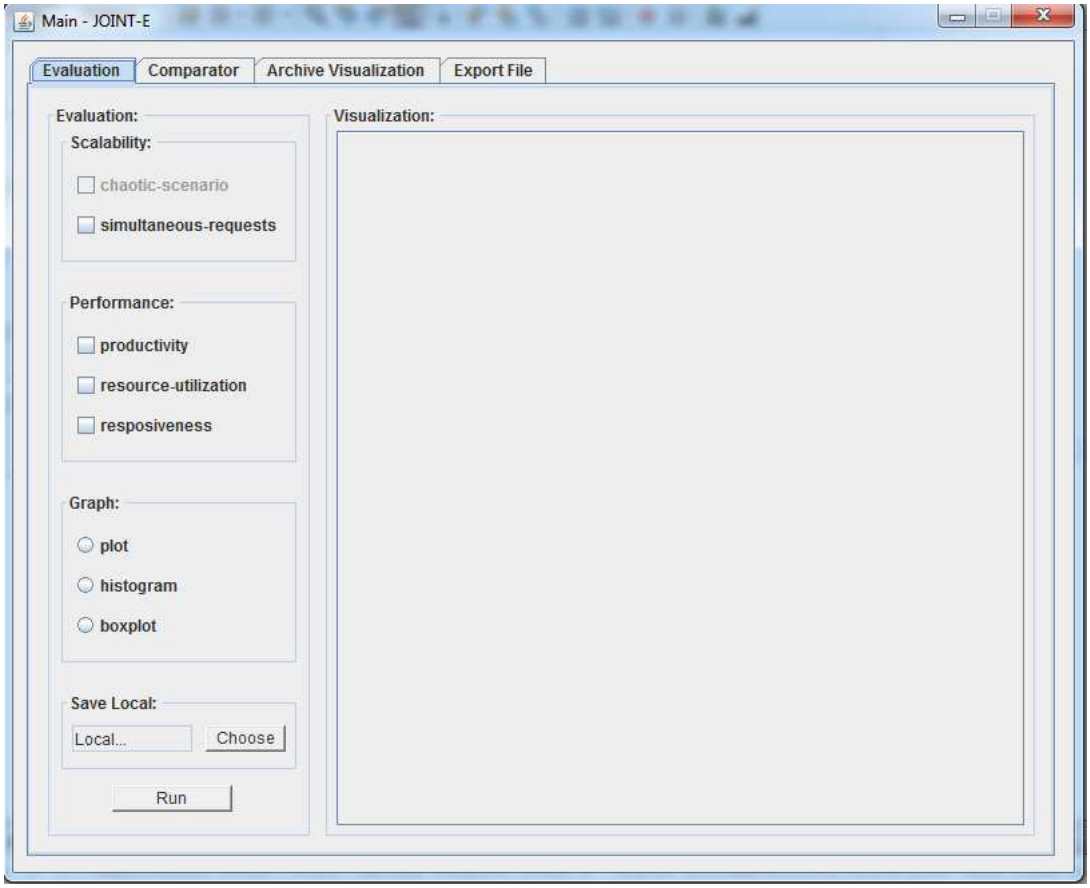

Figura 12 – Interface Gráfica do JOINT-E

3.6 Serviços Gerais

Este módulo disponibiliza serviços básicos que são oferecidos pelo JOINT-E. Estes serviços representam os *frozen spots* do framework. Frisa-se que novos componentes podem ser adicionados a ele, mas normalmente estes componentes são isolados dos outros. Frisa-se ainda que estes serviços foram implementados de forma que pudessem ser facilmente reusáveis (mais detalhes na Seção 3.8), por exemplo, análise estatística. Os serviços disponibilizados por este módulo são:

- **Gerenciamento de Dados**: este serviço gerencia a manipulação e visualização de dados. Através dele é possível exportar os dados em formatos pré-definidos, selecionar as visualizações de dados e carregar análises estatísticas;

- **Avaliação de Desempenho**: por sua vez, este serviço concentra a implementação do plano GQM de desempenho. Dessa maneira, é possível avaliar a API sob o aspecto desempenho, utilizando as diversas questões GQM disponíveis;
- **Análise de Dados**: oferece a análise dos dados obtidos das avaliações das APIs. Este serviço concentra os testes estatísticos, tanto os de normalização quanto os testes de hipóteses;
- **Avaliação de Escalabilidade**: este serviço concentra a implementação do plano GQM de escalabilidade. Dessa maneira, é possível avaliar a API sob o aspecto escalabilidade;
- **Visualização de Dados**: oferece a visualização dos dados obtidos das avaliações das APIs. Este serviço disponibiliza os dados através de vários tipos de gráficos (box-plot, histograma, dentre outros).

3.7 Serviços de Aplicação

Este módulo concentra os serviços de aplicação, onde os mesmos representam os *hot spots* do framework. Para a implementação desse *hot spot* basta que o desenvolvedor implemente a interface explicitada abaixo. Dessa maneira, o Código A.1 apresenta a interface Java *IOperations* a qual representa os serviços de aplicação. Detalhes de como essa interface deve ser implementada seguindo o modelo COSMOS* serão apresentados na Seção 3.11.

Código Fonte 3.1 – Interface Java *IOperations*

```
1  package system.operations.spec.prov;
2
3  /**
4   *
5   * @author Endhe
6   */
7  public interface IOperations {
8
9      public String createObject();
10
11     public Object retrieveObject(String id);
12
13     public boolean updateObject(String id);
14
15     public boolean deleteObject(Object object);
16
17     public void clearAllInstances();
18
19     public int getTotalInstances();
```

```
20 }
```

Abaixo segue a descrição de cada método, onde os mesmos devem ser implementados utilizando a API a qual o desenvolvedor pretende a avaliar.

- **createObject** (linha 9): este método deve criar uma instância na ontologia a qual é manipulada pela API que está sendo avaliada pelo framework. O retorno *String* deve conter o *id* que identifica este objeto na ontologia;
- **retrieveObject** (linha 11): recupera uma instância na ontologia, passando como parâmetro o *id* do objeto;
- **updateObject** (linha 13): a utilização deste método implica na mudança de alguma variável no objeto antes da atualização do mesmo. O retorno booleano serve para indicar se a operação foi executada corretamente.
- **deleteObject** (linha 15): este método remove da ontologia o objeto passado como parâmetro. O retorno booleano serve para indicar se a operação foi executada corretamente.
- **clearAllInstances** (linha 17): este método faz parte do núcleo de operações de gerenciamento básico. O **clearAllInstances** remove todas as instâncias das ontologias.
- **getTotalInstances** (linha 19): retorna o número total de instâncias presentes na ontologia. O mesmo também faz parte do núcleo de operações de gerenciamento básico.

3.8 Implementação

O JOINT-E segue uma implementação baseada em componentes e centrada na arquitetura. Nesse sentido, os componentes foram criados com base no modelo COSMOS*. O COSMOS* usa recursos de linguagem de programação, tais como interfaces, classes e pacotes, e um conjunto de padrões de projeto para representar a arquitetura de software explicitamente no código do programa. Além disso, as orientações definidas pelo modelo COSMOS* também melhoram a modularidade do sistema e a evolução interna dos componentes de software.

Para ilustrar a implementação, a Figura 13 mostra o diagrama de classes do componente *Performance*, que implementa o plano GQM de desempenho.

De acordo com o modelo COSMOS*, o usuário do componente tem acesso apenas ao pacote *spec* e a classe *ComponentFactory*. A interface *IManager* é definida pelo COSMOS* e fornece as operações básicas de gerenciamento do componente, como por exemplo

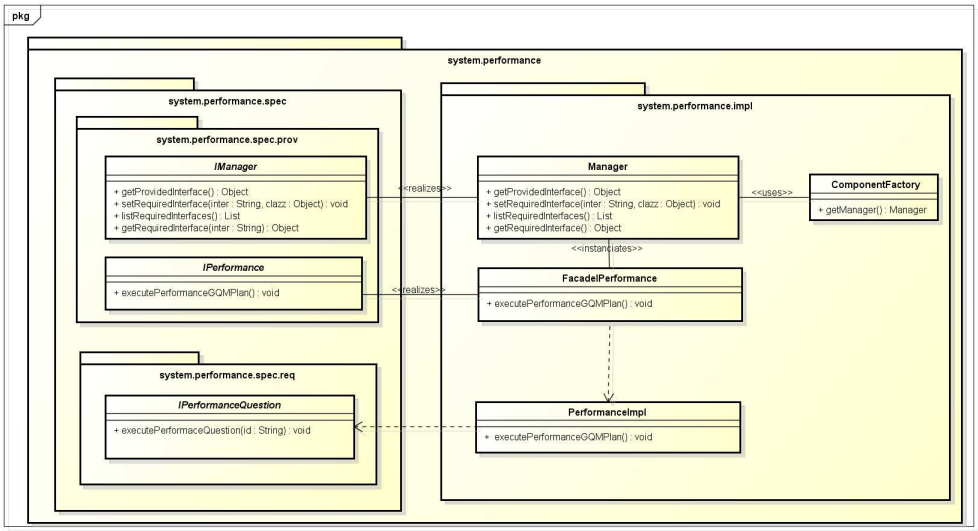

Figura 13 – Componente - *Performance*

acesso as interfaces providas e requeridas. Neste componente a interface *IPerformance* é a interface provida, que contém a implementação do plano GQM de desempenho.

3.9 Fluxo de Execução

O JOINT-E é um framework que possui várias funcionalidades, sendo que podem ser divididas em duas principais: i) avaliar uma API de persistência de ontologias do paradigma OO; ii) comparar as APIs via testes estatísticos. A Figura 3.7 apresenta o fluxo de execução geral para a primeira funcionalidade.

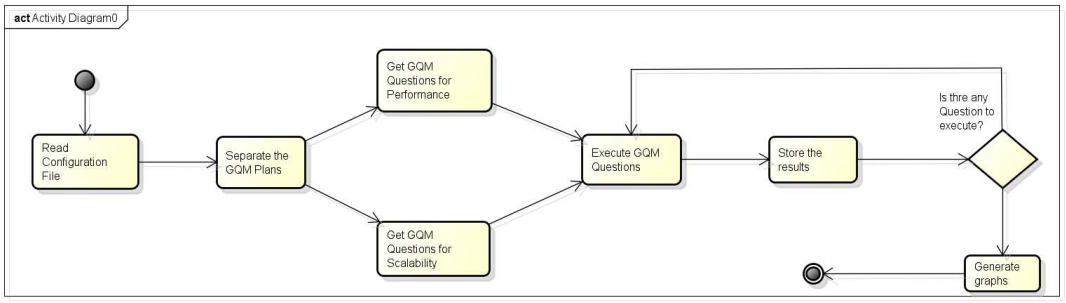

Figura 14 – Fluxo de execução geral do JOINT-E

Primeiramente é feita a leitura do arquivo de configuração, esse arquivo contém informações relacionadas a dados cadastrais e seleção de features (planos GQM selecionados, bem como suas questões; os tipos de gráficos que serão gerados, dentre outras).

Após essa etapa, o JOINT-E seleciona quais planos GQM o usuário do framework quer executar. Logo, é feita a separação das questões no concerne desses planos. Posteriormente, cada questão é executada trinta vezes a fim de se obter uma distribuição que caracterize o resultado da execução da questão GQM.

Por fim, após cada execução é feito o armazenamento dos resultados obtidos. Tal armazenamento ocorre pela camada de persistência. Depois são gerados os gráficos, bem como as exportações (se solicitadas) dos resultados. De posse dos gráficos, o desenvolvedor fica hábil a analisar uma determinada API e decidir se a mesma é adequada para sua aplicação.

3.10 Ferramentas e Persistência

Esta seção tem como objetivo apresentar as ferramentas utilizadas no JOINT-E, bem como o mecanismo de persistência utilizada pelo framework.

As ferramentas são descritas abaixo:

- **R** - A plataforma de análise estatística R foi desenvolvida originalmente por Ross Ihaka e por Robert Gentleman, com o intuito de ser uma linguagem voltada para a análise estatística e, consequentemente, a precisão numérica e com fortes características funcionais (R Development Core Team, 2010). Por este motivo, ela foi utilizada para a análise dos dados e para a geração dos gráficos deste trabalho. A

qualidade numérica desta ferramenta, que foi aferida por (ALMIRON; ALMEIDA; MIRANDA, 2009) é adequada para este framework.

- **JMX Mbeans** - *Java Management Extensions* é uma tecnologia Java que fornece ferramentas para gerenciamento de monitoramento de aplicações, objetos de sistema, dispositivos e redes orientadas a serviço. É uma Interface de Programação de Aplicativos (API), que usa o conceito de agentes, permitindo monitorar elementos da máquina virtual Java (ORACLE, 2014). Essa tecnologia foi utilizada para monitoração de alocação de memória, consumo de CPU e tempo de processamento das requisições.

A camada de persistência do framework foi feita utilizando a seguinte ferramenta:

- **JDOM** - JDOM é um é projeto *open source* desenvolvido em Java para manipulação de arquivos XML. O JDOM integra com Modelo Objeto Documento (DOM) e a *Simple API* para XML (SAX) (COMMUNITY, 2014). O JDOM foi utilizado para manipular os arquivos XML para a persistência do framework.

O módulo de persistência é responsável pelo armazenamento dos dados. Atualmente o JOINT-E utiliza arquivos XML. Dessa maneira, s arquivos são criados dinamicamente, onde para cada questão pertencente a um plano GQM selecionado é criado um arquivo XML. Por exemplo, quando o desenvolvedor seleciona que quer avalia tempo de resposta ($Q_{1.1}$) é criado um arquivo XML corresponde. Vale ressaltar que esses arquivos são utilizados como parâmetro para geração de gráficos e para o módulo de comparação.

3.11 Como instanciar o JOINT-E

O processo de instanciação do JOINT-E é apresentado no Apêndice A.

3.12 Diretrizes do JOINT-E

Esta seção tem como objetivo explanar sobre algumas diretrizes de execução que devem ser consideradas após a instanciação do framework. Tais diretrizes ajudam a reduzir as ameaças à validade da avaliação. Dessa maneira, foram elaboradas baseadas em (GUO; PAN; HEFLIN, 2005; HOST; WOHLIN; RUNESON, 2012)

1. O primeiro ponto que deve ser observado no processo de execução é o estado da máquina, ou seja, é preferível que durante esse processo nenhum outro programa, serviço ou aplicação esteja sendo executado em concomitância. Com isto, é minimizada a variação na medição das unidades de memória e CPU.

2. A máquina virtual do java (do inglês *Java Virtual Machine* (JVM)) possui parâmetros que podem ser configurados antes do processo de execução. Os principais

parâmetros com relação ao framework são os referentes à memória. Dessa maneira, o desenvolvedor precisa definir os limites (máximo e mínimo) de memória da JVM.

3. No processo de comparação de duas APIs, alguns fatores devem ser levados em consideração pelo desenvolvedor a fim de diminuir o risco de resultados tendenciosos entre as APIS. Nesse sentido, os fatores que devem ser observados são:

    - a máquina que foi executada cada API deve possuir a mesma configuração;
    - o estado da máquina deve ser garantido em ambas as execuções;
    - os limites definidos para JVM devem ter sidos os mesmos para as APIs;
    - o modelo de dados utilizado na avaliação deve ser o mesmo.

4. Outro ponto que merece ser analisado pelo desenvolvedor é relacionado ao tamanho da amostra que está sendo analisada, uma vez que o tamanho impacta no processo de avaliação.

4 EXPERIMENTO

Para validar os serviços oferecidos pelo JOINT-E, foram criados três cenários no experimento, onde as principais APIs (Alibaba e Jastor) utilizadas pelos desenvolvedores foram testadas e avaliadas. No experimento foi feita uma análise qualitativa com os resultados estatísticos obtidos, destacando as vantagens e desvantagens de cada API avaliada. É importante frisar que os mesmos resultados obtidos neste trabalho foram compatíveis com os obtidos em trabalhos anteriores.

Por fim, realizou-se também uma pesquisa com desenvolvedores para verificar se o framework JOINT-E oferece informações importantes para tomada de decisão referente à escolha de uma API. Os resultados desta pesquisa constaram que, de fato, o framework oferece meios mais precisos para avaliação das APIs e atende a demanda dos desenvolvedores.

4.1 Cenários

Nesta seção, cada cenário visa ilustrar uma instanciação do JOINT-E utilizando diferentes features e API. Por fim, será apresentada uma discussão sobre os resultados obtidos. Com relação ao ambiente que estes cenários foram executados, a máquina escolhida tinha as seguintes configurações:

- 2.27GHz Intel Core i3 CPU;
- 4GB of RAM; 500GB de HD;
- Sistema operacional Windows 7 Ultimate;
- Java SDK 1.7.0 01; 512MB de memória Heap

4.1.1 Cenário 1: API Alibaba

A API Alibaba é uma coleção de módulos que proveem uma abstração simples do armazenamento RDF feito pelo Sesame (BROEKSTRA; KAMPMAN; HARMELEN, 2002). Nesse sentido, através de uma biblioteca cliente-servidor, a API Alibaba acelera o desenvolvimento, facilita a manutenção e prover um gerador de código Java a partir das ontologias. É importante destacar que a API Alibaba só funciona em conjunto com o armazenamento do Sesame, que por sua vez, oferece vários mecanismos de armazenamento RDF, como PostgreSQL, MySQL, arquivos, dentre outros.

A Figura 15 apresenta a configuração escolhida para este cenário de experimento. Como pode ser visto, o mecanismo de armazenamento escolhido foi o banco de dados PostgreSQL.

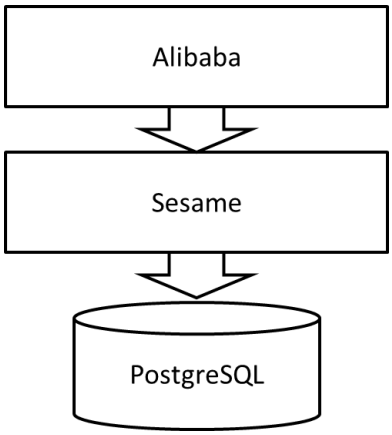

Figura 15 – Cenário de configuração do Alibaba

O código B.1 (Apêncice B) mostra como fica a instanciação do framework utilizando a API Alibaba. É importante notar que há uma abstração do tipo de armazenamento escolhido.

Para este cenário foram escolhidas as seguintes features: *responsiveness, resource utilization* e *simultaneous requests*. Para visualização foram escolhidas as feature *box-plot* e *plot* (gráfico de distribuição). Pode-se observar que todas features são facilmente configuradas através da interface do JOINT-E, como apresentada na Figura 16.

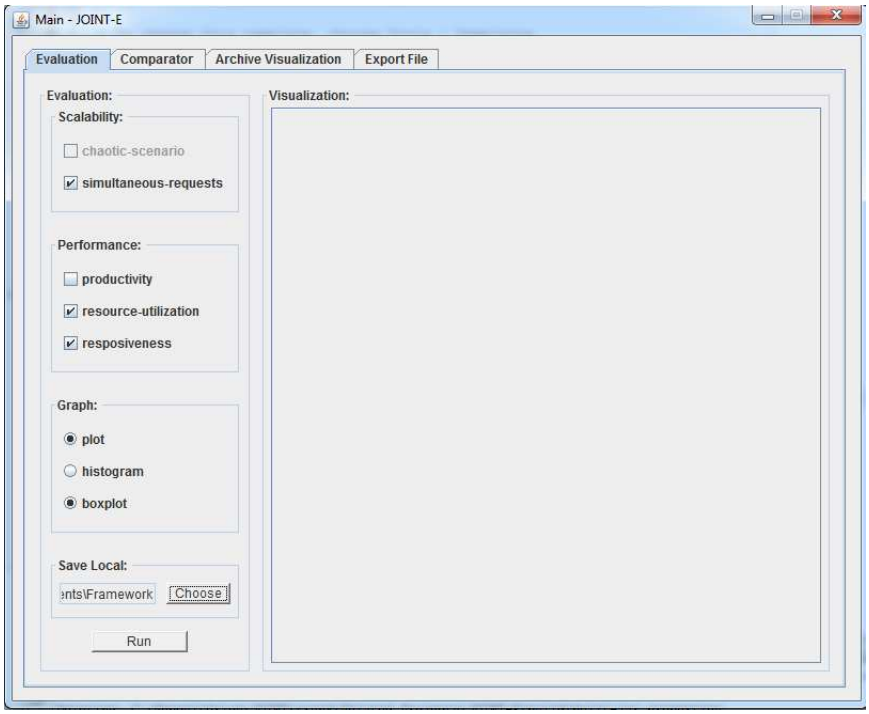

Figura 16 – Seleção de features utilizando a API Alibaba

Para o desenvolvedor é importante ter a formatação do dado bruto, para que seja possível fazer qualquer outro tipo de tratamento que o mesmo julgue necessário. Sendo assim, para cada feature presente em um plano GQM, o JOINT-E gera um arquivo XML.

O Código B.2 apresenta o arquivo após avaliação do aspecto *Responsiveness* e pode ser encontrado no Apêncice B. Através desses arquivos, são gerados os tipos de visualização escolhidos pelos desenvolvedores. Frisa-se que este arquivo pode ser usado como entrada para comparar diferentes APIs, bem como gerar diferentes tipos de visualizações que não foram previamente selecionados.

As figuras 17 e 18 apresentam os resultados obtidos com relação à capacidade de resposta da API Alibaba, levando em consideração cada operação do CRUD. Pode-se observar que tiveram suas medidas máximas perto de 6.000ms enquanto que as medidas mínimas ficaram próximas de 1.000ms.

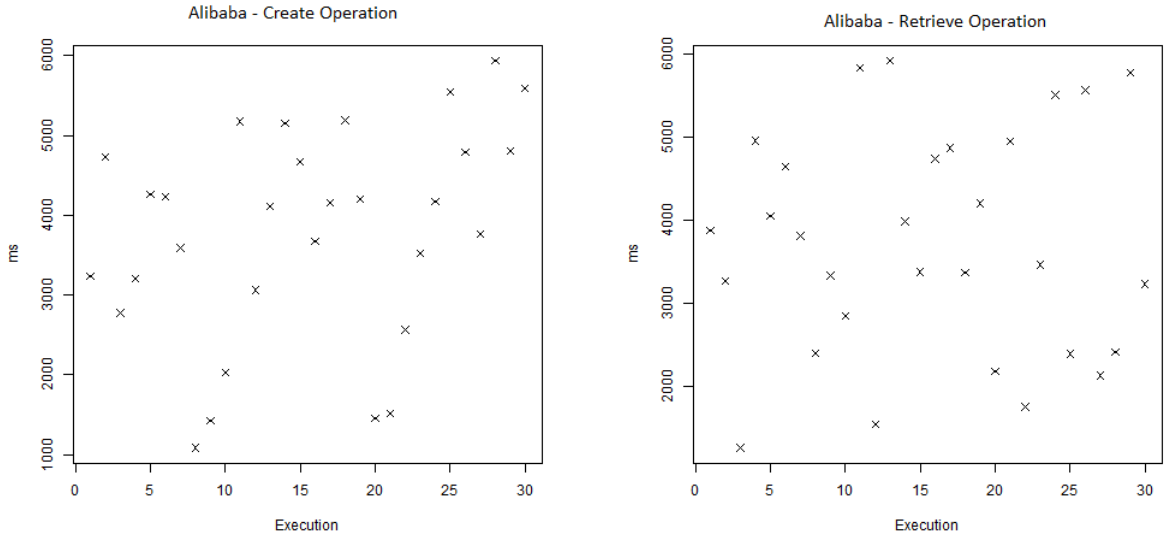

Figura 17 – Distribuição de valores referentes as operações *create* e *retrieve* da API Alibaba - Feature *Responsiveness*

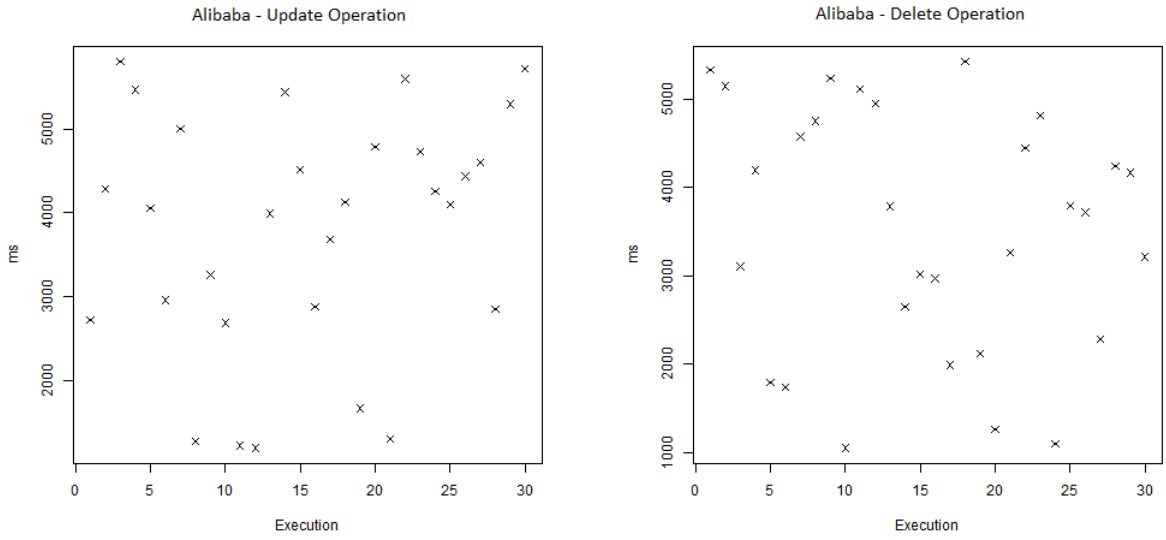

Figura 18 – Distribuição de valores referentes as operações *update* e *delete* da API Alibaba - Feature *Responsiveness*

Por sua vez, as figuras 19 e 20 apresentam *box-plots* que sumarizam os resultados referentes ao consumo de recurso das execuções das operações CRUD. Neste caso, são apresentados gráficos referentes ao uso de memória, contudo também são gerados gráficos com relação ao uso de CPU.

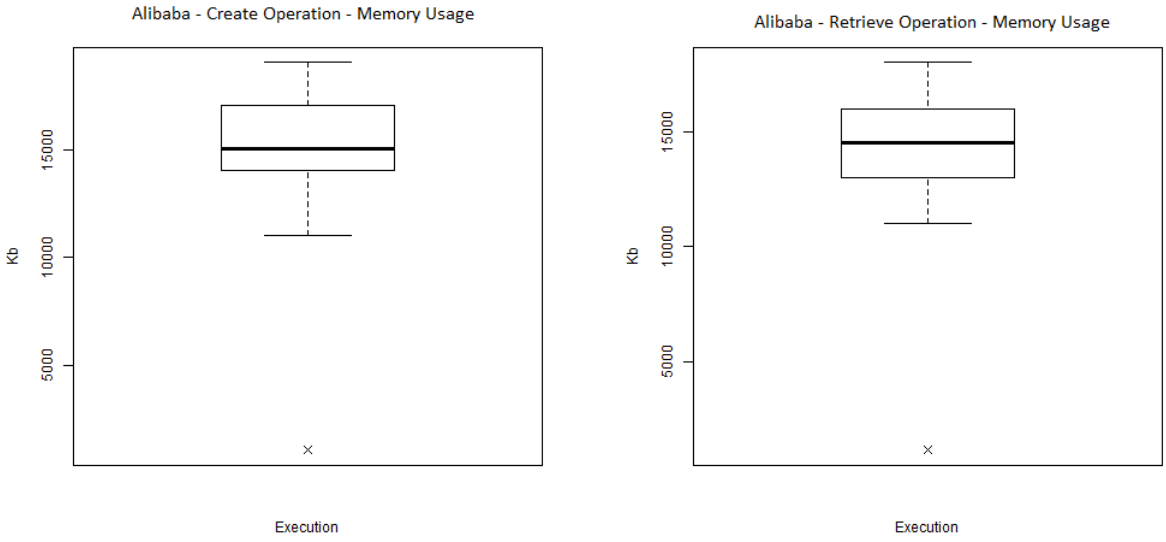

Figura 19 – Box-plots referentes as operações *create* e *retrieve* da API Alibaba - Feature *Resource Utilization*

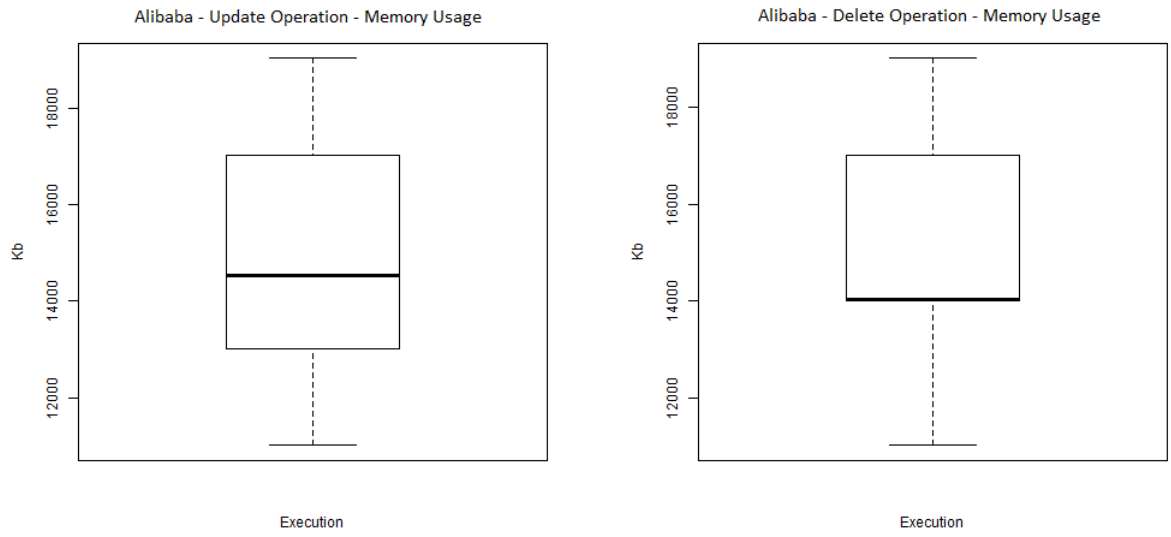

Figura 20 – Box-plots referentes as operações *update* e *delete* da API Alibaba - Feature *Resource Utilization*

Com relação ao critério de avaliação de escalabilidade, a Figura 21 apresenta os valores obtidos para este critério. Como apresentado na Seção 3.1.2 para cada execução é armazenado o número máximo de requisições suportadas naquela execução, onde esse processo é repetido 30 vezes. Como pode ser visto, a API Alibaba suportou aproximadamente 90 requisições enquanto que seu valor mínimo foi de 1 requisição.

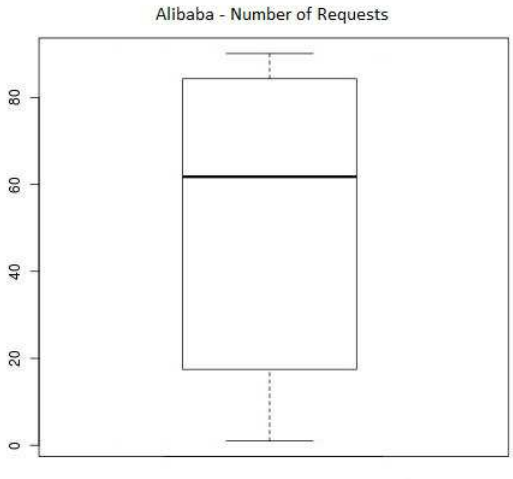

Figura 21 – Box-plot referente a Escalabilidade da Alibaba - Feature *Simultaneous Requests*

4.1.2 Cenário 2

O segundo cenário do experimento consiste em fazer uso da API Jastor. Jastor é um gerador de código de Java que produz JavaBeans a partir de ontologias descritas em

OWL. Assim como a Alibaba, a Jastor também precisa trabalhar em conjunto com um mecanismo de armazenamento RDF, que neste caso é o Jena. Jena é um framework que além de armazenar RDF, oferece um conjunto de ferramentas que podem ser acopladas. É importante ressaltar que o framework Jena oferece vários tipos de armazenamento RDF, como, PostreSQL, MySQL, armazenamento em memória, dentre outros.

A Figura 22 apresenta a configuração escolhida para este cenário de experimento. Assim como no cenário 1, também foi escolhido como mecanismo de armazenamento o PostgreSQL.

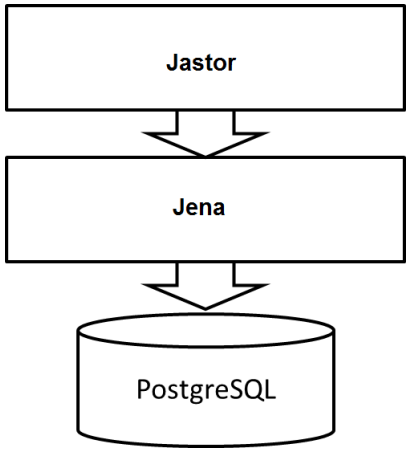

Figura 22 – Cenário de configuração da API Jastor

Continuando com o processo de instanciação do framework, também foi necessário implementar a classe *OperationsImpl*, utilizando a API Jastor. As features selecionadas para este cenário foram *responsiveness, resource utilization* e *simultaneous requests*. Além disso, para visualização foi escolhida apenas a feature *box-plot*. A configuração dessas features pode ser vista na Figura 23.

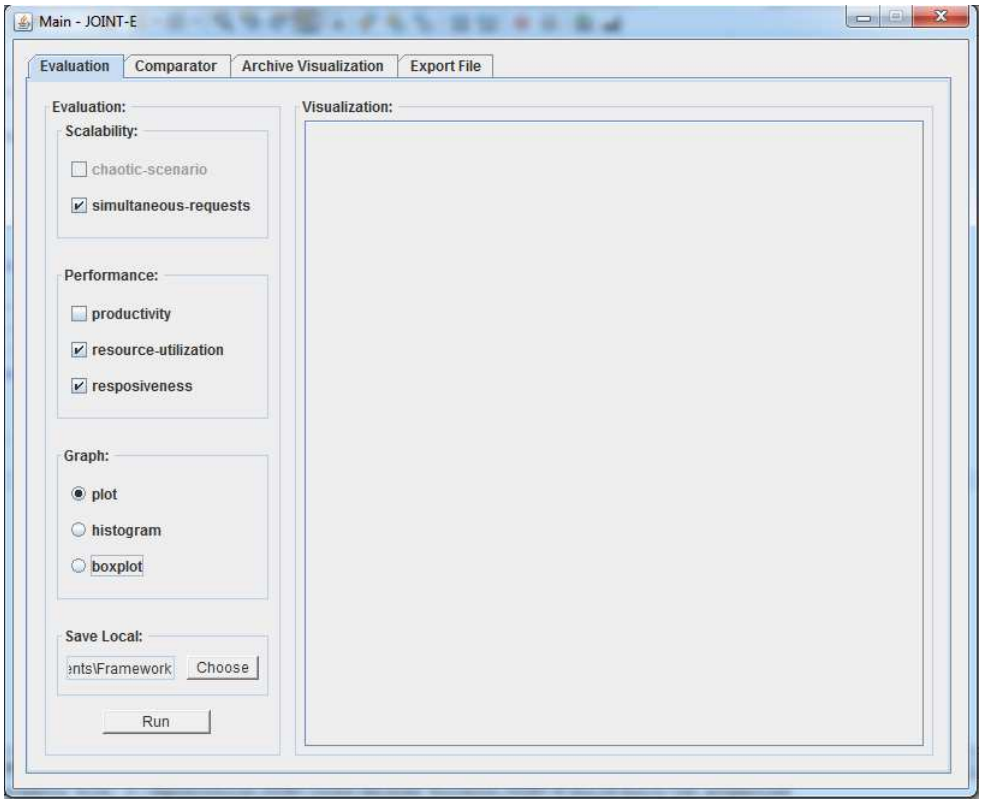

Figura 23 – Cenário de configuração da API Jastor

Os resultados obtidos com relação a capacidade de resposta da API Jastor podem ser vistos através das figuras 24 e 25. Ao contrário da API Alibaba, a Jastor apresentou um resultado melhor ficando a maioria de suas execuções abaixo de 1000ms.

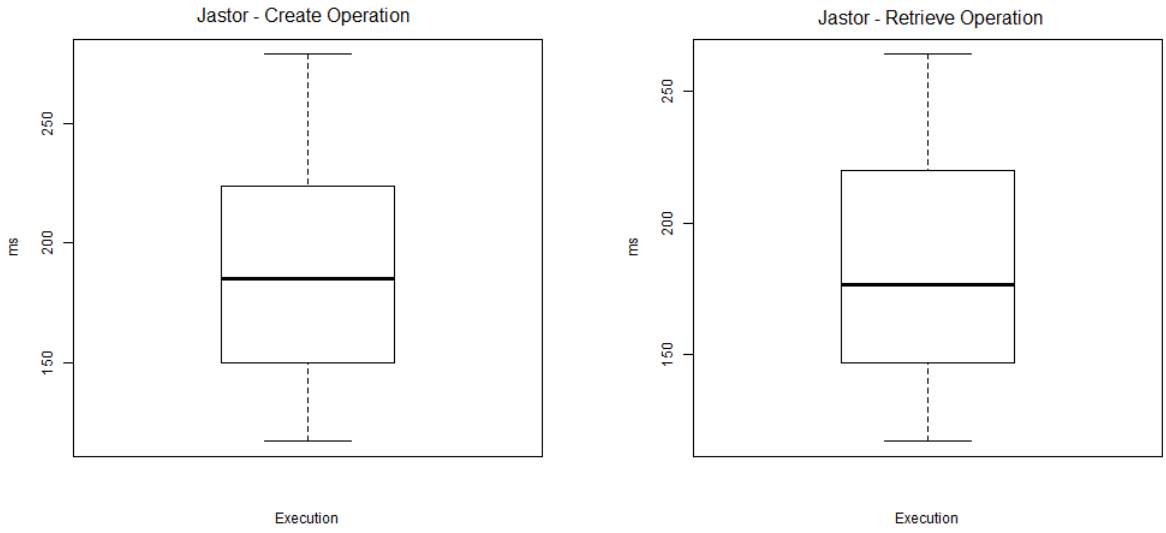

Figura 24 – Box-plots referentes as operações *create* e *retrieve* da API Jastor - Feature *Responsiveness*

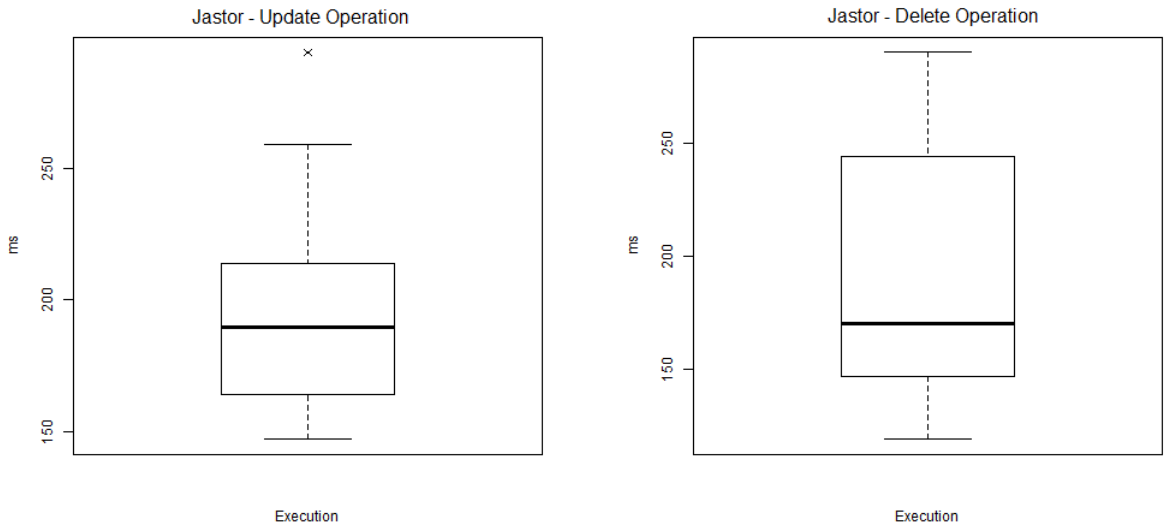

Figura 25 – Box-plots referentes as operações *update* e *delete* da API Jastor - Feature *Responsiveness*

Analisando os resultados da API Jastor com relação ao consumo de memória, observa-se que variaram entre 5000kb e 10000kb. Esses resultados podem ser vistos nas figuras 26 e 27.

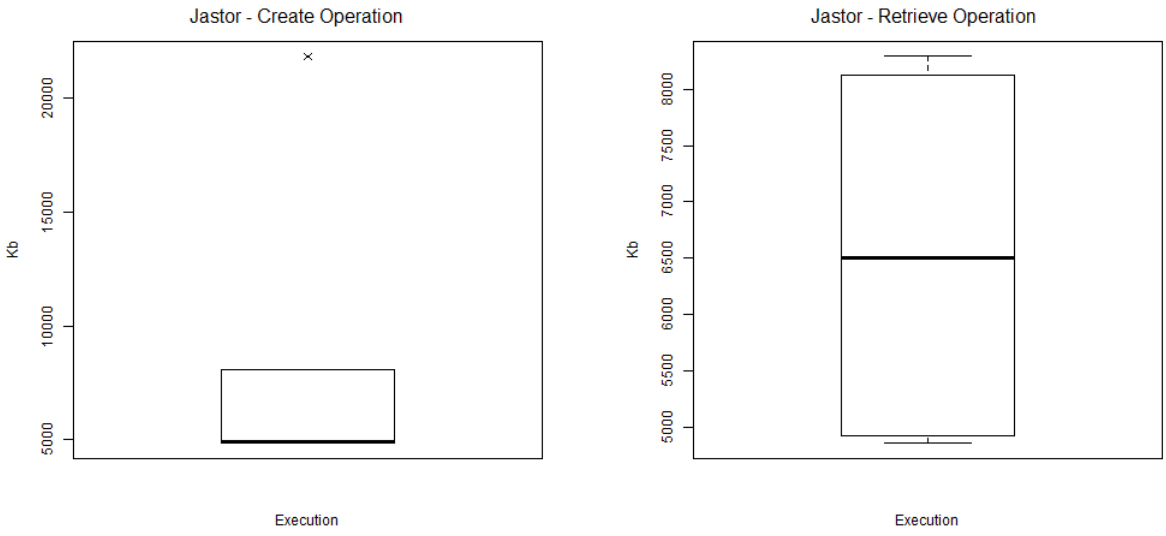

Figura 26 – Box-plots referentes as operações *create* e *retrieve* da API Jastor - Feature *Resource Utilization*

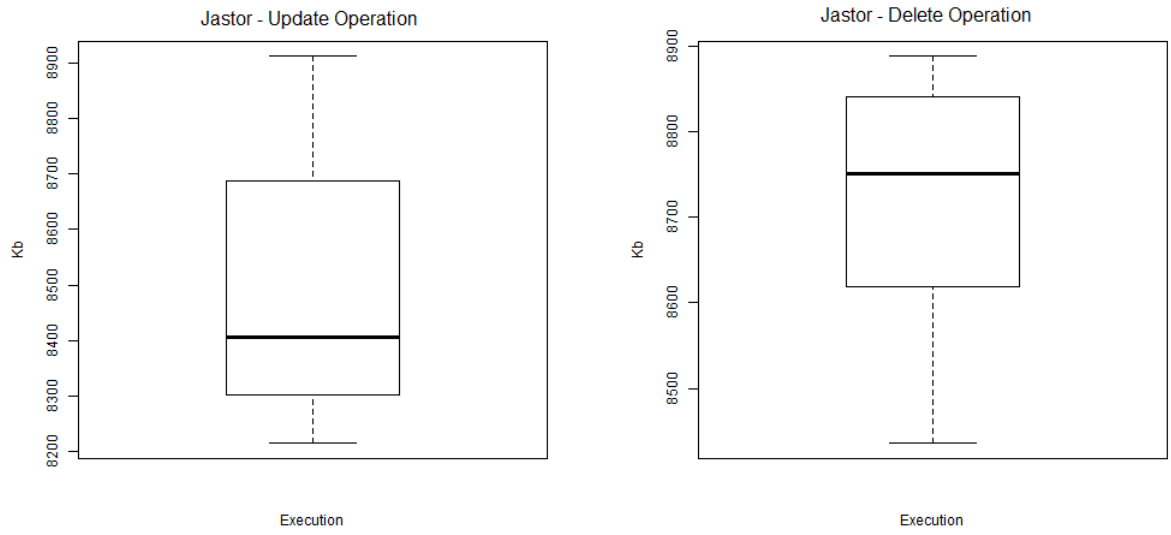

Figura 27 – Box-plots referentes as operações *update* e *delete* da API Jastor - Feature *Resource Utilization*

Por fim, o critério de avaliação de escalabilidade é apresentado na Figura 28. Para o Jastor os valores máximos e mínimos são aproximadamente 70 e 80. Como pode ser visto esses valores são semelhantes ao Alibaba e serão discutidos no próximo cenário.

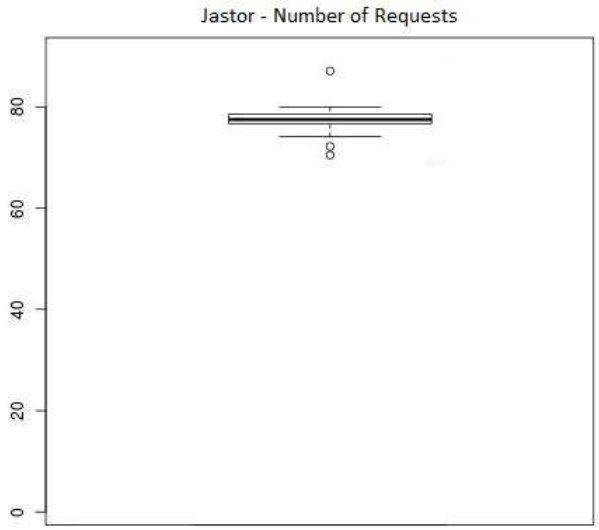

Figura 28 – Box-plot referente a Escalabilidade da Jastor - Feature *Simultaneous Requests*

4.1.3 Cenário 3

Apesar de avaliar uma API separadamente possa dar indicações de quão adequada ela é para o desenvolvedor, o mesmo precisa comparar os resultados de outras avaliações a fim

de tomar a decisão baseada em resultados mais consistentes. Neste sentido, o JOINT-E prover o serviço de comparação de resultados através de testes estatísticos. Frisa-se que o JOINT-E sumariza o resultado para o desenvolvedor, deixando toda lógica de testes de hipóteses de maneira transparente.

A comparação é feita de maneira simples basta o desenvolvedor carrega através da interface gráfica dois arquivos de saída de avaliação. É importante destacar que o framework só compara arquivos de avaliação da mesma questão GQM e só pode ser feita uma comparação dois a dois. Assim, a Figura 29 apresenta a comparação entre a capacidade de resposta (feature *Responsiveness*) de cada API. Como pode ser visto, a API Jastor se mostra superior, pois quanto menor o tempo de resposta melhor para o desenvolvedor.

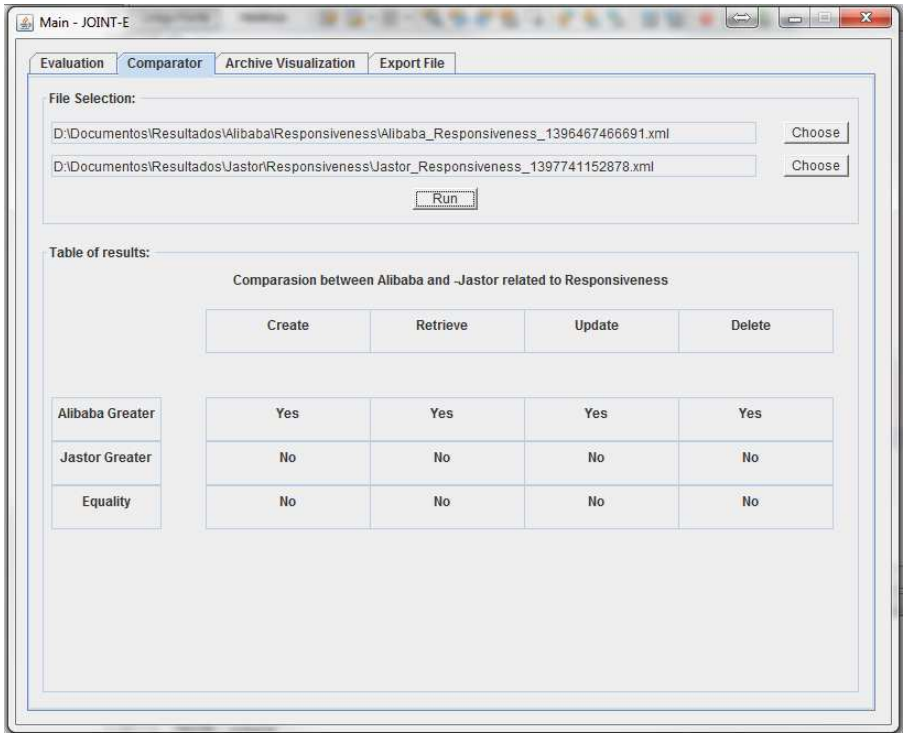

Figura 29 – Comparação entre as capacidades de respostas das APIs Alibaba e Jastor

O outro aspecto ilustrado neste cenário é a comparação entre APIs com relação ao consumo de recursos, sendo apresentado o aspecto memória. Assim como na comparação anterior, a API Jastor superou positivamente a API Alibaba, vide Figura 30.

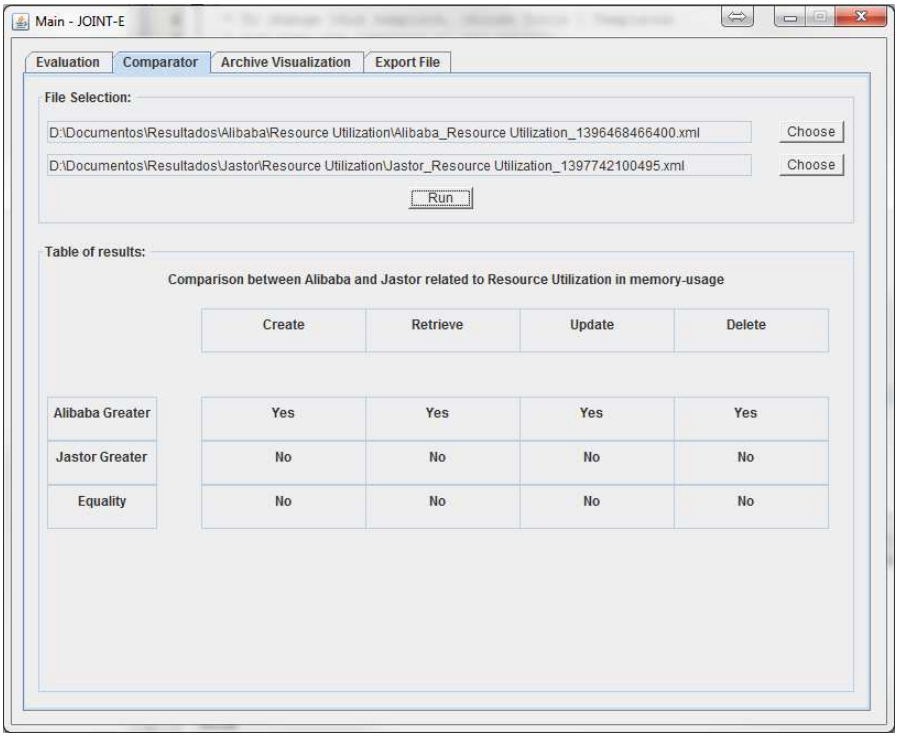

Figura 30 – Comparação entre consumo de memória das APIs Alibaba e Jastor

Ressalta-se que as comparações acima afirmam os indícios dados pelas avaliações individuais das APIs. Onde a API Jastor mostrou ter um melhor desempenho sob essas questões do que a API Alibaba.

Por fim, a Figura 31 apresenta a comparação com relação à escalabilidade. Como foram apresentados nos cenários anteriores, os valores obtidos das APIs Alibaba e Jastor foram bastantes próximos um do outro. Dessa maneira, os testes de hipóteses se mostraram inconclusivos, pois não foi possível refutar nenhuma das hipóteses. Entretanto, analisando os box-plots obtidos é possível observar que a Jastor é mais estável enquanto que a Alibaba varia bastante de acordo com as execuções. Para este caso, é mostrado apenas - (traço) no lugar de “Yes” ou “No”.

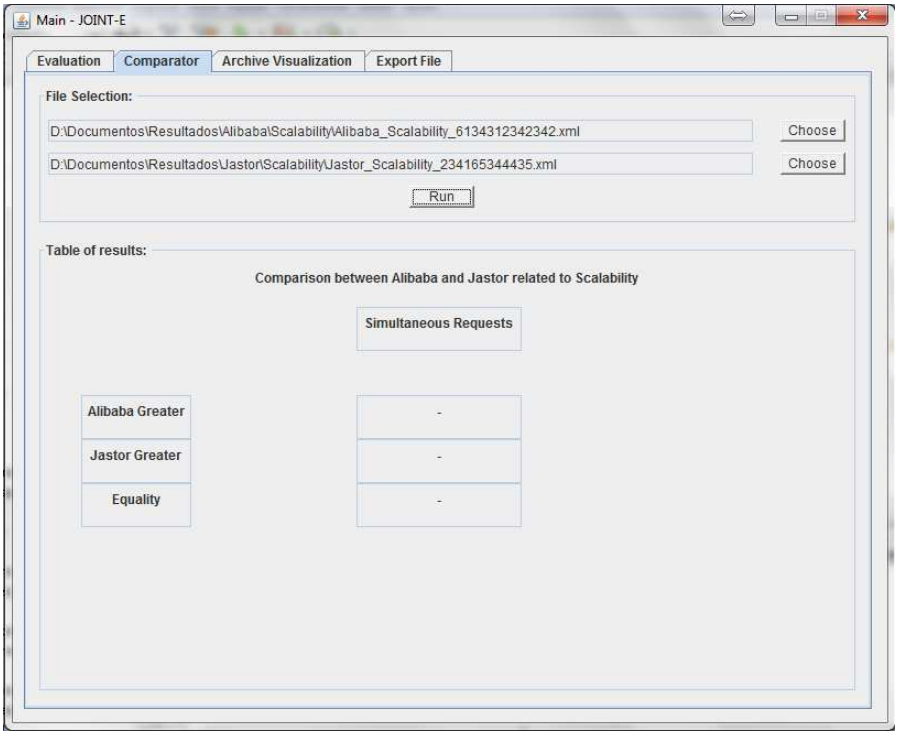

Figura 31 – Comparação entre consumo de memória das APIs Alibaba e Jastor

4.1.4 Discussão sobre os Cenários

Analisando os resultados apresentados nos cenários acima, pode-se observar que a API Jastor superou a API Alibaba em todos os aspectos avaliados, exceto na escalabilidade, pois os resultados se mostram inconclusivos. Uma possibilidade do porquê a API Alibaba obteve um desempenho inferior é que ela utiliza os construtores OWL diretamente nos objetos gerados. Isto facilita a leitura perante o desenvolvedor, contudo pode prejudicar o desempenho, visto que há um aumento de processamento.

Diferentemente da API Alibaba, a API Jastor tem como escopo apenas a manipulação dos objetos, sem se preocupar com as propriedades e as restrições do OWL. Além disso, a Alibaba se encontra em versão beta, enquanto que Jastor está em sua versão estável.

4.2 Pesquisa de Opinião

A fim de obter uma validação diretamente com estes o usuários, foi aplicada uma pesquisa de opinião com questões que vão desde captação do perfil dos usuários até as necessidades mais técnicas. A pesquisa pode ser encontrado no Apêndice C.

4.2.1 Perfil dos Usuários

A pesquisa contou participantes de todo país com diversos níveis de formação indo desde ensino médio a doutores, no total foram 46 participantes. As Figuras 32 e 33 mostra a distribuição dos participantes nos estados brasileiros, bem como o nível de formação.

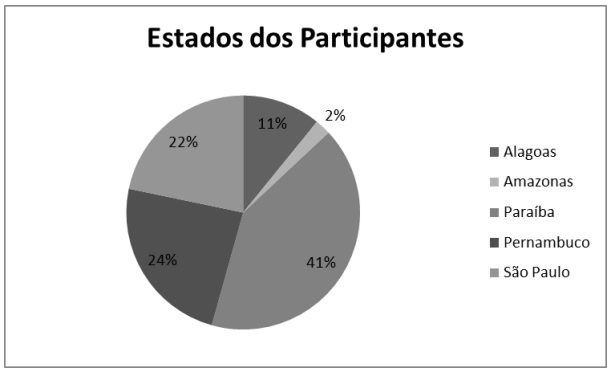

Figura 32 – Estados dos partipantes

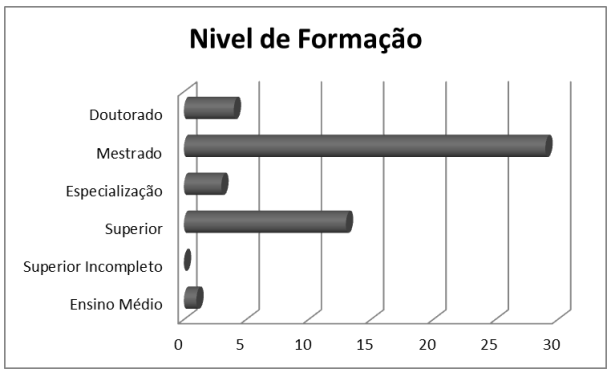

Figura 33 – Nível de formação dos partipantes

Com relação à atuação no campo da pesquisa científica ou no mercado de trabalho, é apresentado nas Figuras 34 e 35 que a maioria dos participantes vem da pesquisa científica, havendo também aqueles que atuam também no mercado de trabalho.

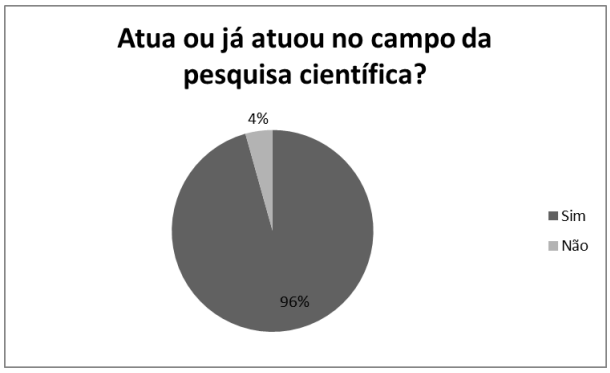

Figura 34 – Atuação na pesquisa científica

Além disso, a pesquisa de opinião buscou extrair outros aspectos dos participantes. Dessa maneira, foram levantadas as linguagens de programação utilizadas e há quanto tempo os participantes programam (Figuras 36 e 37. É possível observar que a linguagem de programação mais utilizada é a Java e a experiência na programação supera os 4 anos.

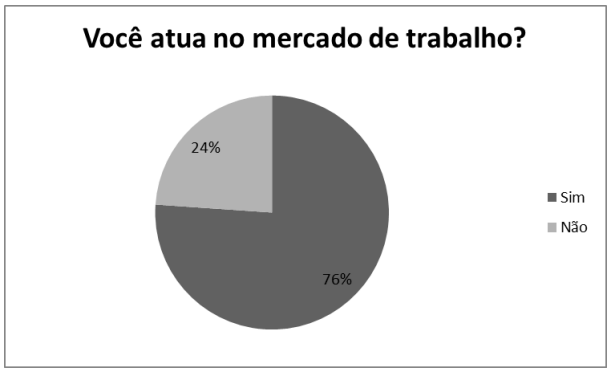

Figura 35 – Atuação no mercado de trabalho

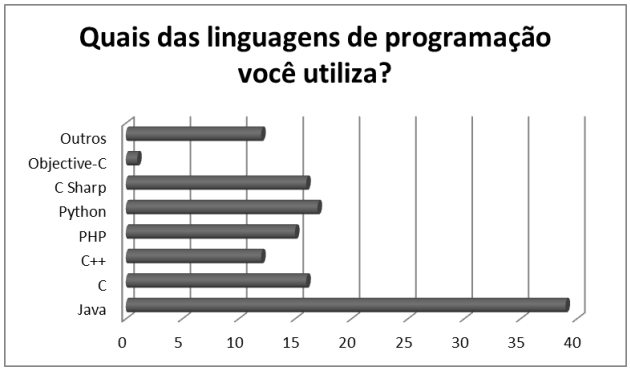

Figura 36 – Linguagens utilizadas

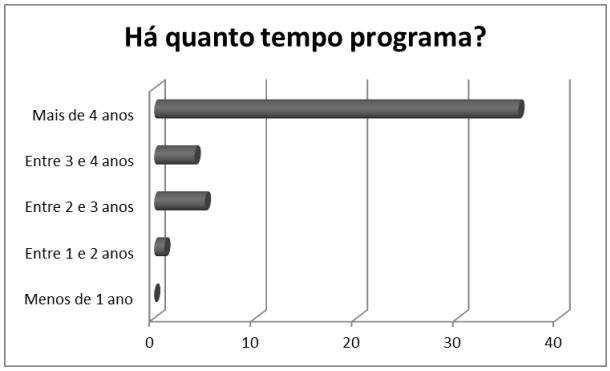

Figura 37 – Experiência de programação

É importante destacar que aspectos como a frequência a qual os desenvolvedores tomam decisão sobre a escolha das APIs, bem como, as características que são levadas em consideração na avaliação precisam ser averiguadas pela pesquisa de opinião. Sendo assim, as Figuras 38 e 39 apresenta as respostas deste questionamento. Pode-se notar que a maioria dos participantes frequentemente precisam tomar decisões sobre a escolha da API e que os principais aspectos levados em consideração é flexibilidade (uso e integração), desempenho e escalabilidade.

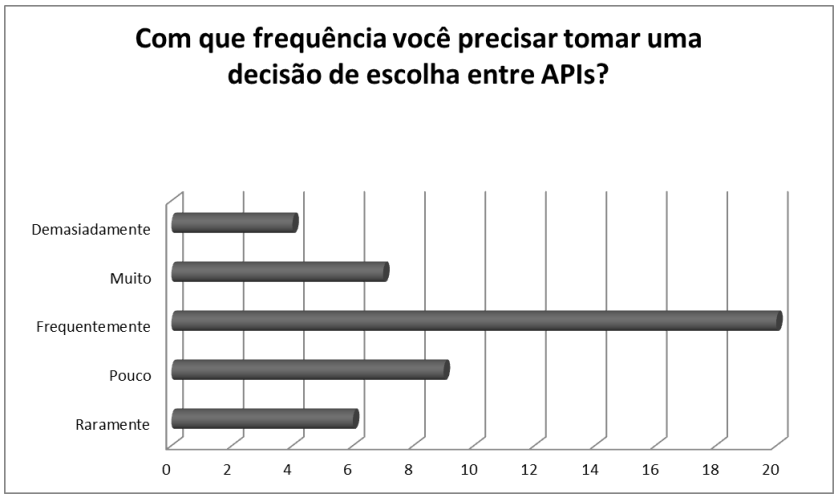

Figura 38 – Frequência de avaliação

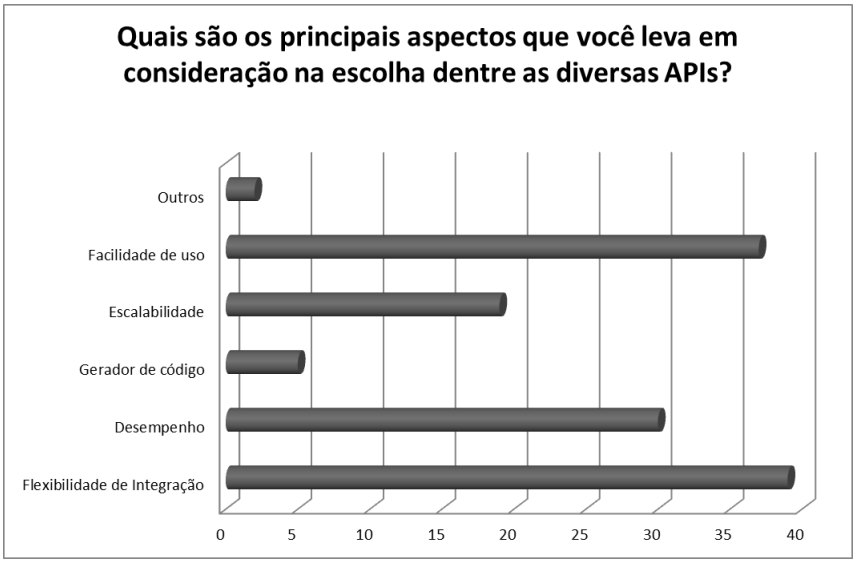

Figura 39 – Aspectos das APIs

4.2.2 Aspectos de Avaliação

A pesquisa contou também com o objetivo de verificar como os desenvolvedores observam a avaliação de desempenho e escalabilidade, indo desde aspectos, como características das APIs à tipos de visualização de dados.

Dessa maneira, a Figura 40 apresenta a necessidade sentida pelos desenvolvedores na avaliação de desempenho sob o aspecto tempo de resposta. Como pode ser visto, mais de 20 desenvolvedores consideram como muito importante este tipo de avaliação. Além disso, mais de 10 consideram extremamente importante. Por sua vez, a Figura 41 apresenta os resultados da pesquisa de opinião com relação a utilização de recurso, onde números para importante e extremamente importante são os mesmos do gráfico anterior.

Outro aspecto avaliado foi como os desenvolvedores enxergam avaliação de escalabilidade. De acordo com a Figura 42 veem como números de requisicação simultâneas

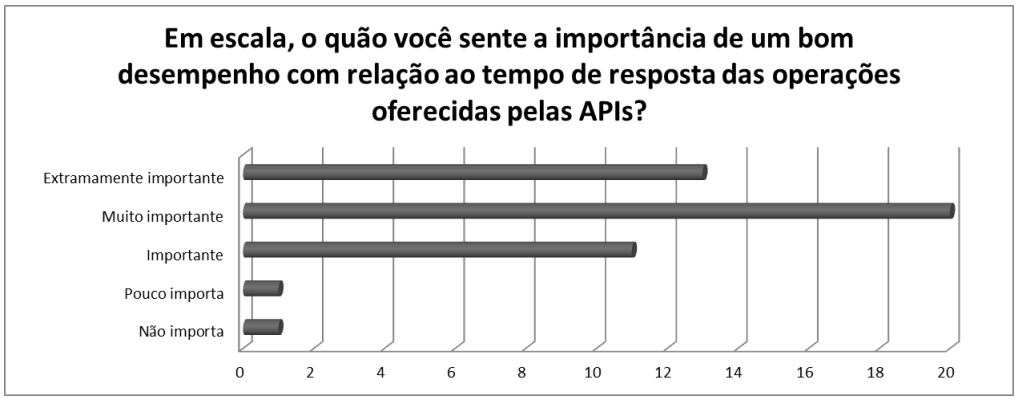

Figura 40 – Necessidade de avaliação de desempenho relacionada ao tempo de resposta

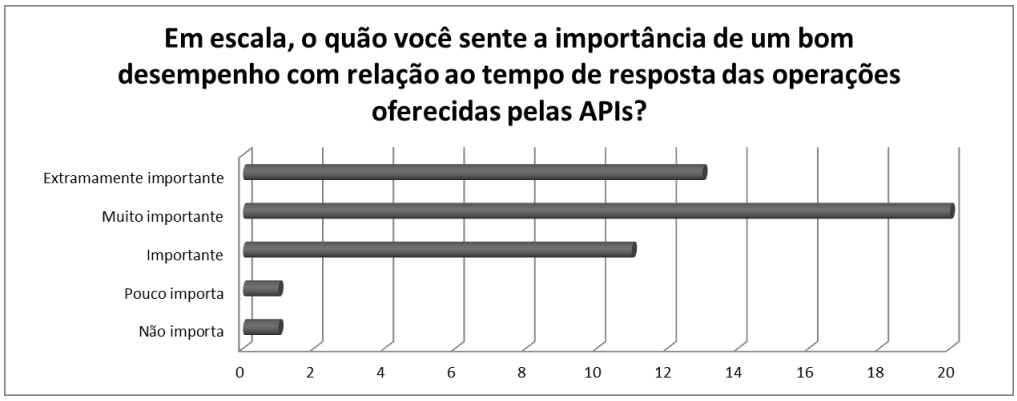

Figura 41 – Necessidade de avaliação de desempenho relacionada à utilização de recurso

suportadas pela APIs como o principal aspecto de avaliação. É importante destacar que throughtput e simulação de cenários caóticos também foram levados em consideração pelos desenvolvedores.

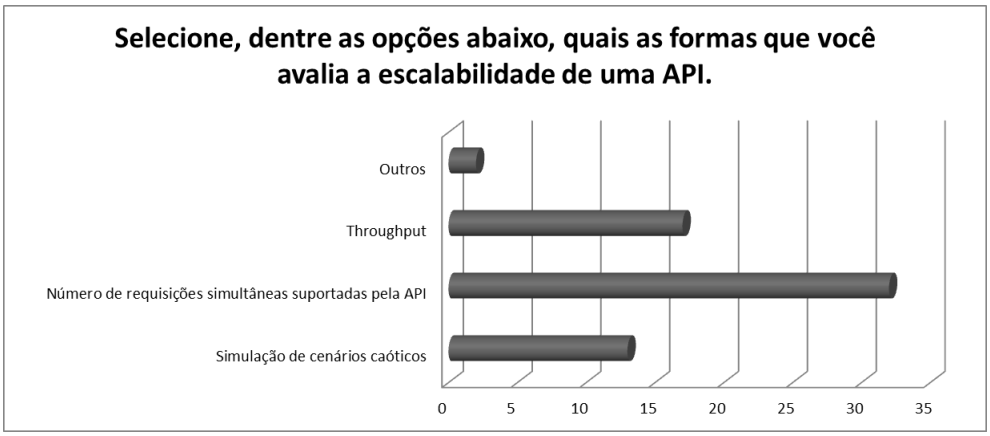

Figura 42 – Aspectos da avaliação de Escalabilidade

Além disso, os desenvolvedores também foram capazes de explanar a necessidade de um mecanismo o qual os ajudasse a avaliar as APIs. Sendo assim, a Figura 43 mostra que 28 desenvolvedores consideram essa necessidade como de muito ou de extrema importância.

Além disso, mais de 10 desenvolvedores a consideram como de necessidade mediana. Com relação à visualização de dados as mais interessantes para os desenvolvedores são gráficos em linha e tabelas. Entretanto, formas que enfatizam a distribuição de dados também são vistas como interessantes, tais como: box-plot e histogramas (Figura 44).

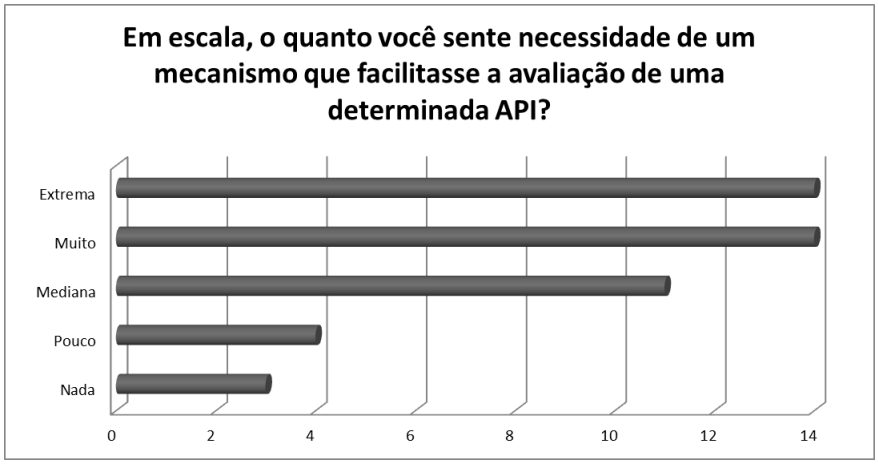

Figura 43 – Necessidade de um mecanismo de avaliação

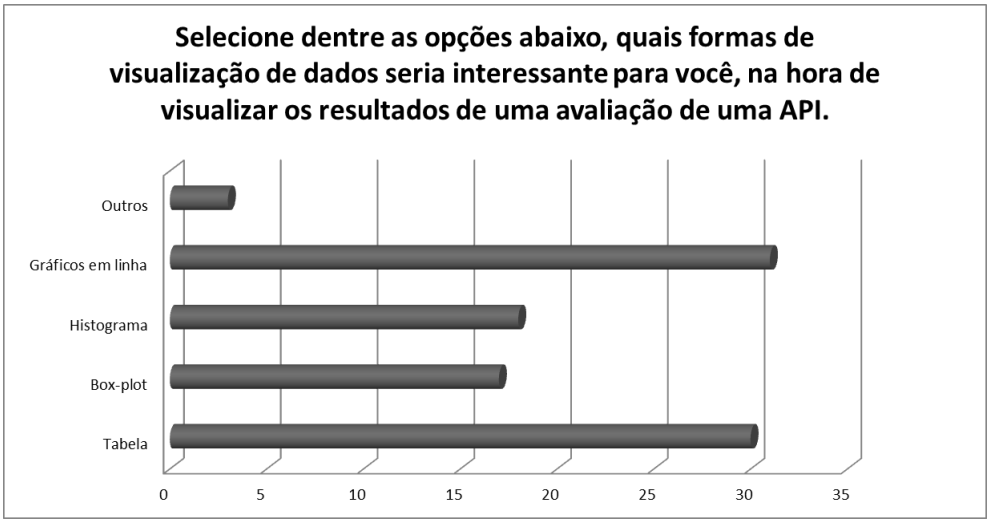

Figura 44 – Tipos de visualização de dados

Por fim, a Figura 45 apresenta os resultados apresentados pelos desenvolvedores com relação a necessidade de um mecanismo que os ajudasse a comparar os resultados. Como pode ser visto, mais de 80% dos entrevistados consideram de mediana à extrema importância a necessidade desse mecanismo. Sendo assim, todos os aspectos do questionário estão alinhados com os fornecidos pelos JOINT-E.

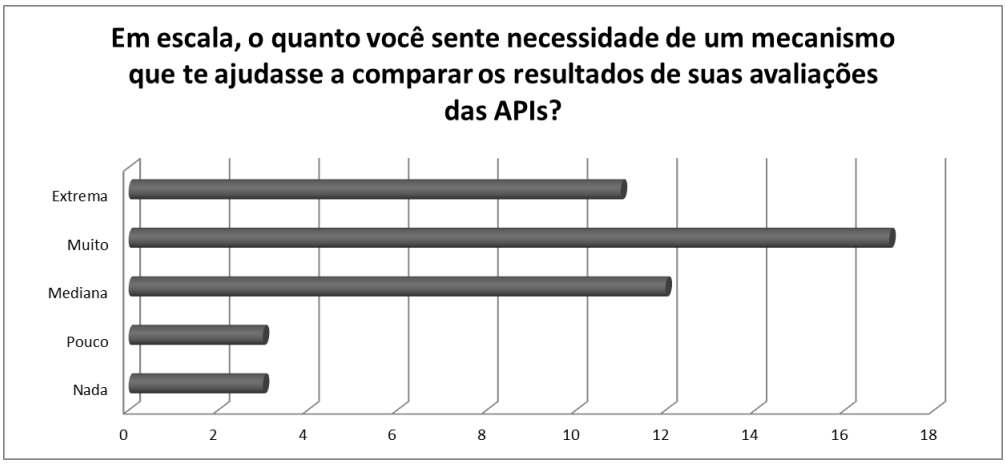

Figura 45 – Necessidade de um mecanismo de comparação

5 CONCLUSÃO

Este trabalho teve como objetivo apresentar um framework centrado na arquitetura para avaliar o desempenho e a escalabilidade das APIs de manipulação de ontologias do paradigma POO. Dessa maneira, o JOINT-E funciona na camada de alto nível de abstração e cobre fatores desde acesso aos dados, instanciação, atualização e recuperação e remoção. Em comparação com trabalhos anteriores, o trabalho proposto foi além do demais, considerando mais aspectos, como desempenho, escalabilidade ambos direcionados as APIs POO.

A viabilidade do framework foi validada sob duas perspectivas: i) criação de cenários para ilustrar o funcionamento do framework; e ii) validação diretamente com os desenvolvedores através de uma pesquisa que foi aplicada. Os resultados do experimento mostraram que Jastor é significantemente melhor que Alibaba. Por sua vez, o resultado da pesquisa mostrou que o framework atende uma necessidade por parte dos desenvolvedores.

Além disso, os cenários apresentados no experimento, também podem ser úteis aos desenvolvedores, fornecendo um guia para ajuda-los na tomada decisão dentre as diversas APIs e cenários de configuração disponíveis. As principais contribuições do trabalho foram os reposicionamentos das métricas de avaliação para as APIs do paradigma POO, a concepção e desenvolvimento do framework JOINT-E e os resultados obtidos no experimento.

Entretanto, esta dissertação ainda possui algumas limitações, como avaliar as APIs POO em outros aspectos, por exemplo, flexibilidade e integração das APIs, suporte às consultas, documentação, clareza no mapeamento entre ontologia-objeto.

Para os trabalhos futuros, espera-se aumentar a cobertura de fatores de avaliação das APIs, validar o framework com mais desenvolvedores e aplica-lo em um cenário de mercado. Com relação à comparação estatística das APIs, se tem como trabalho futuro incluir a possibilidade do desenvolvedor avaliar várias APIs paralelamente.

Durante o período do mestrado, outros trabalhos foram desenvolvidos paralelamente, dentre eles destacam-se: i) (HOLANDA et al., 2013b) publicado no *journal Expert Systems with Applications* (qualis A1); ii) (DERMEVAL et al., 2013) publicado no *28th Annual ACM Symposium* (qualis A1); e iii) (HOLANDA et al., 2013a) publicado no *IEEE/WIC/ACM International Joint Conferences on Web Intelligence* (WI) *and Intelligent Agent Technologies* (IAT) (qualis B1).

REFERÊNCIAS

ABADI, D. J. et al. Scalable semantic web data management using vertical partitioning. In: *Proceedings of the 33rd international conference on Very large data bases*. Vienna, Austria: VLDB Endowment, 2007. p. 411–422.

ALEXAKI, S. et al. On storing voluminous rdf descriptions: The case of web portal catalogs. In: *4th International Workshop on the Web and Databases*. Santa Barbara, California, USA: WebDB, 2001.

ALMIRON, M.; ALMEIDA, E. S.; MIRANDA, M. The reliability of statistical functions in four software packages freely used in numerical computation. *Brazilian Journal of Probability and Statistics*, Special Issue on Statistical Image and Signal Processing, p. 107–119, 2009.

BASILI, V.; CALDEIRA, G.; ROMBACH, H. D. The goal question metric approach. In: *Encyclopedia of Software Engineering*. Department of Computer Science, University of Maryland, College Park: Wiley, 1994.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific American*, v. 84, p. 34–43, 2001.

BOLEY, H.; TABET, S.; WAGNER, G. Design rationale of ruleml: A markup language for semantic web rules. In: *Proceedings of Second International Workshop*. Sanibel Island, Florida, USA: RuleML, 2001. p. 381–401.

BROEKSTRA, J.; KAMPMAN, A.; HARMELEN, F. van. Sesame: A generic architecture for storing and querying rdf and rdf schema. In: *The Semantic Web - ISWC 2002*. New York, USA: Springer Berlin / Heidelberg, 2002. p. 54–68.

CARDOSO, J.; LYTRAS, M. D. *Semantic Web Engineering in the Knowledge Society*. Agia Paraskevi, Greece: IGI Global, 2009.

CASTRO, S. *Ontologia*. Rio de Janeiro, Brasil: Zahar, 2008.

CHEN, F. et al. An architecture-based approach for component-oriented development. In: *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*. Oxford, England: IEEE, 2002. p. 450–455.

CHESSMAN, J.; DANIELS, J. *UML Components*. Indiana, USA: Addison-Wesley, 2000.

COMMUNITY, J. *Java-based Document Object Model for XML (JDOM)*. 2014. Http://www.jdom.org/. Último acesso em: Fevereiro de 2014.

DERMEVAL, D. et al. On the use of metamodeling for relating requirements and architectural design decisions. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2013. p. 1278–1283.

DEVEDZIC, V. *Semantic Web and Education*. New York, USA: Springer, 1 edition., 2006.

FAYAD, M.; SCHMIDT, D.; JOHNSON, R. *Building application frameworks: object-oriented foundations of framework design*. New York, USA: John Wiley & Sons, Inc., 1999.

GARCIA-CASTRO, R.; GOMEZ-PEREZ, A. Guidelines for benchmarking the performance of ontology management apis. In: *Semantic Web - ISWC 2005, Proceedings*. Galway, Ireland: Springer Berlin / Heidelberg, 2005.

GARLAN, D.; SHAW, M. *An introduction to software architecture*. Pittsburgh, USA, 1994.

GAYARD, L. A.; RUBIRA, C. M. F.; GUERRA, P. A. de C. *COSMOS*: a COmponent System MOdel for Software Architectures*. Campinas, Brasil, 2008.

GROVE, M. *Empire: RDF & SPARQL Meet JPA*. 2010. Available from https://github.com/mhgrove/Empire.

GRUBER, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, v. 5, p. 199–220, 1993.

GUARINO, N. Formal ontology and information systems. In: *Proceedings of FOIS'98*. Trento, Italy: IOS Press, 1998. p. 3–15.

GUO, Y.; PAN, Z.; HEFLIN, J. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 3, p. 158 – 182, 2005.

HARMELEN, F. van. Semantic web research anno 2006: Main streams, popular fallacies, current status and future challenges. In: *Cooperative Information Agents X, Proceedings*. Edinburgh, UK: Springer Berlin / Heidelberg, 2006.

HENDLER, J. et al. Web science: an interdisciplinary approach to understanding the web. *Communication ACM*, v. 51, p. 60–69, 2008.

HEPP, M. et al. *Ontology management: semantic web, semantic web services, and business applications*. New York, USA: Springer Berlin / Heidelberg, 2008.

HERTEL, A.; BROEKSTRA, J.; STUCKENSCHMIDT, H. Rdf storage and retrieval systems. In: *Handbook on Ontologies*. Munich, Germany: Springer Berlin Heidelberg, 2009. p. 489–508.

HOLANDA, O. et al. Towards an agent-based approach for automatic generation of researcher profiles using multiple data sources. In: *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), International Joint Conferences on*. Atlanta, USA: IEEE/WIC/ACM, 2013. v. 3, p. 163–166.

HOLANDA, O. et al. Joint: Java ontology integrated toolkit. *Expert Systems with Applications*, v. 40, p. 6469 – 6477, 2013.

HOST, M.; WOHLIN, C.; RUNESON, P. *Experimentation in Software Engineering*. Norwell, USA: Springer, 2012.

JOHNSON, R. Components, frameworks, patterns. *Communications of the ACM*, v. 40, p. 10–17, 1997.

JOHNSON, R.; FOOTE, B. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1(2), p. 22–35, 1988.

KANG, K. C. et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Pittsburgh, USA, 1990.

KOSKIMIES, K. Towards architecture-oriented programming environments. In: CITESEER. *ASERC Workshop on Software Architecture*. Colorado, USA, 2001.

LEIGH, J. *Alibaba*. 2011. Available from http://www.openrdf.org/doc/alibaba/2.0-beta14/.

MAGKANARAKI, A. et al. Benchmarking rdf schemas for the semantic web. In: *Semantic Web - ISWC 2002*. Sardinia, Italy: Springer Berlin / Heidelberg, 2002.

MELLO, R. F. L. *RetriBlog: Um Framework Centrado na Arquitetura para Criação de Blog Crawlers*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2011.

MENASCE, D. A.; ALMEIDA, V. A.; DOWDY, L. W. *Performance by Design: Computer Capacity Planning by Example*. Tennessee, USA: Pearson Education, Inc., 2004.

ORACLE. *Java Management Extensions (JMX)*. 2014. Http://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html. Último acesso em: Fevereiro de 2014.

R Development Core Team. *The R Project for Statistical Computing*. Vienna, Austria, 2010. Disponível em: <http://www.r-project.org>.

SAUVE, J. P. *Projeto de Software Orientado a Objeto*. 2000. Http://www.dsc.ufcg.edu.br/ jacques/cursos/map/html/frame/oque.htm. Último acesso em Fevereiro de 2014.

SOMMERVILLE, I. *Software Engineering (7th Edition) (International Computer Science Series)*. Edinburgh, UK: Addison Wesley, 2004.

SZEKELY, B.; BETZ, J. *Jastor: Typesafe, ontology driven RDF access from java*. 2009. Available from http://jastor.sourceforge.net.

SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*. Zurich, Switzerland: Addison-Wesley, 1998.

TERZIYAN, V.; KONONENKO, O. Semantic web enabled web services: State-of-art and industrial challenges. In: *Web Services - ICWS - Europe 2003, Proceedings*. Karlsruhe, Germany: Springer Berlin / Heidelberg, 2003.

THEOHARIS, Y.; CHRISTOPHIDES, V.; KARVOUNARAKIS, G. Benchmarking database representations of rdf/s stores. In: *Semantic Web - ISWC 2005, Proceedings*. Galway, Ireland: Springer Berlin / Heidelberg, 2005.

TRIOLA, M. F. *Introdução à Estatística*. Rio de Janeiro, Brasil: Ed. LTC, 1999.

WENZEL, K. *KOMMA: An Application Framework for Ontology-based Software Systems*. 2010. IOS Press (http://semantic-web-journal.com/sejp/page/node/89).

YOU-SHENG, Z.; YU-YUN, H. Architecture-based sw process model. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 28, n. 2, p. 16, 2003.

ZIMMERMANN, M. *Owl2Java - A Java Code Generator for OWL*. 2014. Available from http://www.incunabulum.de/projects/it/owl2java.

ZIMMERMANN, R. et al. Scalability evaluation of the yima streaming media architecture. *Software-Practice and Experience*, WILEY-BLACKWELL, v. 35, n. 4, p. 345–359, APR 10 2005.

APÊNDICE A – INSTANCIANDO O JOINT-E

Os passos abaixo ilustram o processam de instanciação do framework JOINT-E, indo desde configuração do projeto até a interação com a interface gráfica.

1. Fazer o download do JOINT-E no link:
   http://sourceforge.net/projects/jointnees/files/JOINT_E.zip/download

2. Importar para o Netbeans:

   a) Abra o Netbeans;

   b) Clique em file->open project;

   c) Escolha o projeto do JOINT-E.

3. Abaixo será apresentada a configuração completa da plataforma R para o sistema operacional Windows. Porém, em outros sistemas operacionais, o processo para configuração é semelhante. Frisa-se que é necessário ter um ambiente Java já preparado e com todas as suas variáveis de ambiente, devidamente configuradas. A configuração do R se dar da seguinte maneira:

   a) Instalação do R:
   Link para download: <www.vps.fmvz.usp.br/CRAN/bin/windows/base/>

   b) Após ter o R instalado, é necessário efetuar o download do **rJava**. Com o console do R aberto, cole o seguinte comando: **install.packages ("Rjava")**, em seguida escolha algum espelho CRAN para deseja efetuar o download. Com o arquivo **rJava** baixado, basta extraí-lo e coloca-lo dentro da biblioteca do diretório de instalação do R. Por exemplo, "$C:\ProgramFiles\R\R-3.0.2\library$";

   c) Com as duas etapas concluídas, é necessário ajustar as variáveis de ambiente para R_HOME contendo o diretório de instalação, e em seguidas adicionar os caminhos das DLLs que se encontram em rJava no Path do Windows. Ex.: jir.dll, rJava.dll. Como exemplificado abaixo:

   $JAVA\_HOME: C:\ProgramFiles\Java\jdk1.7.0\_25$
   $R\_HOME: C:\ProgramFiles\R\R-3.0.1$
   PATH (part):
   $(\ldots)C:\ProgramFiles\Java\jdk1.7.0\_25\jre\bin;$
   $C:\ProgramFiles\Java\jdk1.7.0\_25\jre\bin\server;$
   $C:\ProgramFiles\R\R-3.0.1\bin\x64;$
   $C:\ProgramFiles\R\R-3.0.1\library\rJava\jri\x64$

4. Este passo de instalação do framework é opcional, uma vez que o desenvolvedor pode apenas utilizar o módulo de comparação, não sendo necessário assim, implementar o componente *Operações*. A Figura 46 apresenta a estrutura do componente *Operações*, vale ressaltar que essa estrutura é fornecida pelo framework.

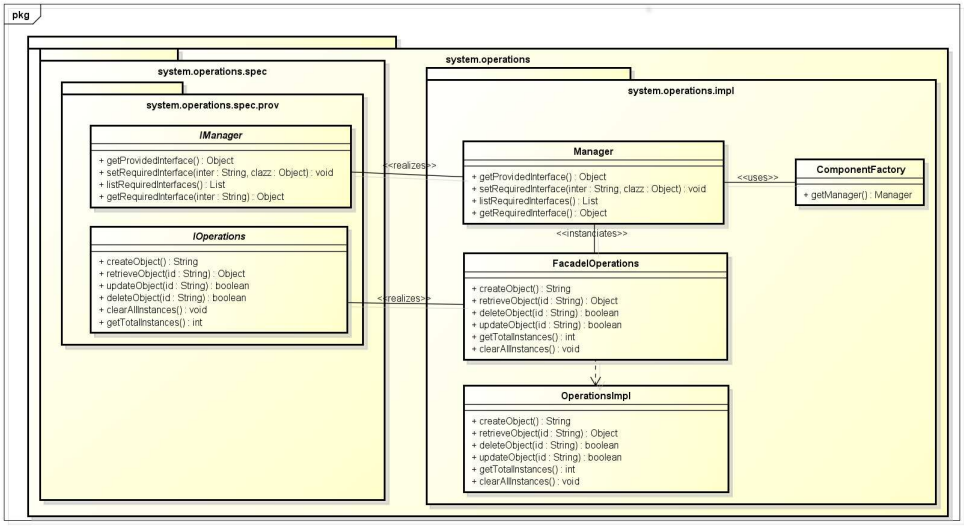

Figura 46 – Diagrama de Pacotes - Componente *Operações*

Dessa maneira, o desenvolvedor apenas precisa implementar os métodos da classe *OperationsImpl* a qual é apresenta abaixo. A lógica de implementação de cada método está apresentada na Seção 3.7.

Código Fonte A.1 – Classe Java *OperationsImpl*

```
1  package system.operations.impl;
2
3  /**
4   *
5   * @author Endhe
6   */
7  public class OperationsImpl{
8
9      @Override
10     public String createObject() {
11         throw new UnsupportedOperationException("Not supported yet.");
12     }
13
14     @Override
15     public Object retrieveObject(String id) {
16         throw new UnsupportedOperationException("Not supported yet.");
17     }
18
19     @Override
20     public boolean updateObject(String id) {
```

```
21          throw new UnsupportedOperationException("Not supported yet.");
22      }
23
24      @Override
25      public boolean deleteObject(Object object) {
26          throw new UnsupportedOperationException("Not supported yet.");
27      }
28
29      @Override
30      public void clearAllInstances() {
31          throw new UnsupportedOperationException("Not supported yet.");
32      }
33
34      @Override
35      public int getTotalInstances() {
36          throw new UnsupportedOperationException("Not supported yet.");
37      }
38  }
```

5. A interação do desenvolvedor com o framework se dar através do componente JOINT-E. Esse componente tem um papel de agregador de funções de interação com o framework.

APÊNDICE B – CÓDIGOS

B.1 Instanciação do JOINT-E usando a Alibaba

O código B.1 mostra como fica a implementação do hot-spot do JOINT-E utilizando a API Alibaba.

Código Fonte B.1 – Classe Java *OperationsImpl*

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package system.operations.impl;
6
7  import java.util.ArrayList;
8  import java.util.HashSet;
9  import java.util.List;
10 import java.util.Set;
11 import java.util.UUID;
12 import java.util.logging.Level;
13 import java.util.logging.Logger;
14 import joint.codegen.nsf716b93d.University;
15 import org.openrdf.model.Resource;
16 import org.openrdf.model.URI;
17 import org.openrdf.repository.Repository;
18 import org.openrdf.repository.RepositoryException;
19 import org.openrdf.repository.http.HTTPRepository;
20 import org.openrdf.repository.object.ObjectConnection;
21 import org.openrdf.repository.object.config.ObjectRepositoryFactory;
22 import org.openrdf.result.Result;
23
24 /**
25  *
26  * @author Endhe
27  */
28 public class OperationsImplAlibaba {
29
30     private String urlRepository = "http://localhost:8080/openrdf-sesame/
          repositories/r";
31     private String ontologyURI = "http://swat.cse.lehigh.edu/onto/univ-
          bench-instances.owl";
32     private Repository repository;
33     private Class classe = University.class;
34     private boolean control = true;
35     private List<String> createdIDs;
36
```

```
37      public OperationsImplAlibaba() {
38          init();
39          createdIDs = new ArrayList<>();
40      }
41
42      void init() {
43          if (control) {
44              repository = new HTTPRepository(urlRepository);
45              try {
46                  repository.initialize();
47              } catch (RepositoryException ex) {
48                  Logger.getLogger(OperationsImplAlibaba.class.getName())
49                                  .log(Level.SEVERE, null, ex);
50              }
51              control = false;
52          }
53      }
54
55      public String createObject() throws Exception {
56          init();
57          ObjectRepositoryFactory factory = new ObjectRepositoryFactory();
58          ObjectConnection connection = factory.createRepository(repository).
                getConnection();
59          connection.setAutoCommit(false);
60          String id = generateID();
61          createdIDs.add(id);
62          Object ob = connection.getObjectFactory().createObject(this.
                ontologyURI + "#" + id);
63
64          try {
65              // Saves the object in the repository
66              connection.addObject(ob);
67              ob = connection.addDesignation(ob, classe);
68              connection.commit();
69              connection.close();
70              return id;
71          } catch (Exception e) {
72              connection.rollback();
73              connection.close();
74              throw new Exception();
75          }
76      }
77
78      public Object retrieveObject(String id) throws Exception {
79          init();
80          ObjectRepositoryFactory factory = new ObjectRepositoryFactory();
81          ObjectConnection connection = factory.createRepository(repository).
```

```
            getConnection();
82          connection.setAutoCommit(false);
83
84          Object ob = null;
85
86          // Creates the instance URI
87          URI instance = connection.getValueFactory().createURI(
88                  this.ontologyURI + "#" + id);
89
90          // Retrieves the desired instance with the URI and the .class
91          ob = connection.getObject(this.classe, instance);
92
93          connection.commit();
94          connection.close();
95          return ob;
96
97      }
98
99      private Set<String> generateSetName() {
100         Set<String> set = new HashSet<>();
101         set.add(generateID());
102         return set;
103     }
104
105     public boolean updateObject(String id) throws Exception {
106         try {
107             init();
108             ObjectRepositoryFactory factory = new ObjectRepositoryFactory()
                    ;
109             ObjectConnection connection = factory.
110                         createRepository(repository).getConnection();
111             connection.setAutoCommit(false);
112
113             URI instance = connection.getValueFactory().createURI(
114                     this.ontologyURI + "#" + id);
115
116             University u = (University) connection.getObject(this.classe,
                    instance);
117             u.setName(generateSetName());
118             connection.commit();
119             connection.close();
120             return true;
121         } catch (Exception e) {
122             return false;
123         }
124     }
125
```

```
126     public boolean deleteObject(String id) throws Exception {
127         try {
128             ObjectRepositoryFactory factory = new ObjectRepositoryFactory()
                    ;
129             ObjectConnection connection = factory.createRepository(
                    repository).getConnection();
130             connection.setAutoCommit(false);
131 
132             URI instance = connection.getValueFactory().createURI(
133                     this.ontologyURI + "#" + id);
134 
135             connection.remove(instance, null, null);
136             connection.remove((Resource) null, null, instance);
137             connection.commit();
138             connection.close();
139             return true;
140         } catch (Exception e) {
141             return false;
142         }
143     }
144 
145     public void clearAllInstances() {
146         try {
147             ObjectRepositoryFactory factory = new ObjectRepositoryFactory()
                    ;
148             ObjectConnection connection = factory.createRepository(
                    repository).getConnection();
149             connection.setAutoCommit(false);
150 
151             for (String id : createdIDs) {
152 
153                 URI instance = connection.getValueFactory().createURI(
154                         this.ontologyURI + "#" + id);
155 
156                 connection.remove(instance, null, null);
157                 connection.remove((Resource) null, null, instance);
158             }
159             connection.commit();
160             connection.close();
161         } catch (Exception ex) {
162             Logger.getLogger(OperationsImplAlibaba.class.getName()).log(
                    Level.SEVERE, null, ex);
163         }
164     }
165 
166     public int getTotalInstances() {
167         try {
```

```
168             ObjectRepositoryFactory factory = new ObjectRepositoryFactory()
                    ;
169             ObjectConnection connection = factory.createRepository(
                    repository).getConnection();
170             connection.setAutoCommit(false);
171             Result<?> r = (Result<?>) connection.getObjects(this.classe);
172             List<?> instances = r.asList();
173             return instances.size();
174         } catch (Exception ex) {
175             Logger.getLogger(OperationsImplAlibaba.class.getName()).log(
                    Level.SEVERE, null, ex);
176             return 0;
177         }
178     }
179
180     private String generateID() {
181         UUID uuid = UUID.randomUUID();
182         return uuid.toString();
183     }
184 }
```

B.2 Arquivo XML

O Código B.2 apresenta o arquivo após a execução da feature *responsiveness*.

Código Fonte B.2 – Arquivo XML correspondente a execução da feature responsiveness

```
1  <model.Evaluation>
2    <questionName>Responsiveness</questionName>
3    <operations>
4      <model.Operation>
5        <values>
6          <float>3236.0</float>
7          <float>4733.0</float>
8          <float>2773.0</float>
9          <float>3203.0</float>
10         <float>4265.0</float>
11         <float>4232.0</float>
12         <float>3589.0</float>
13         <float>1083.0</float>
14         <float>1421.0</float>
15         <float>2029.0</float>
16         <float>5176.0</float>
17         <float>3063.0</float>
18         <float>4112.0</float>
19         <float>5150.0</float>
20         <float>4665.0</float>
21         <float>3676.0</float>
```

```
22            <float>4159.0</float>
23            <float>5188.0</float>
24            <float>4202.0</float>
25            <float>1451.0</float>
26            <float>1515.0</float>
27            <float>2565.0</float>
28            <float>3522.0</float>
29            <float>4170.0</float>
30            <float>5545.0</float>
31            <float>4791.0</float>
32            <float>3763.0</float>
33            <float>5937.0</float>
34            <float>4800.0</float>
35            <float>5586.0</float>
36          </values>
37          <operationName>create</operationName>
38          <measuredUnit>ms</measuredUnit>
39        </model.Operation>
40        <model.Operation>
41          <values>
42            <float>5330.0</float>
43            <float>5149.0</float>
44            <float>3107.0</float>
45            <float>4195.0</float>
46            <float>1791.0</float>
47            <float>1736.0</float>
48            <float>4573.0</float>
49            <float>4754.0</float>
50            <float>5232.0</float>
51            <float>1048.0</float>
52            <float>5115.0</float>
53            <float>4951.0</float>
54            <float>3788.0</float>
55            <float>2653.0</float>
56            <float>3016.0</float>
57            <float>2972.0</float>
58            <float>1990.0</float>
59            <float>5427.0</float>
60            <float>2122.0</float>
61            <float>1265.0</float>
62            <float>3266.0</float>
63            <float>4448.0</float>
64            <float>4811.0</float>
65            <float>1102.0</float>
66            <float>3794.0</float>
67            <float>3720.0</float>
68            <float>2282.0</float>
```

```
69            <float>4245.0</float>
70            <float>4165.0</float>
71            <float>3213.0</float>
72          </values>
73          <operationName>delete</operationName>
74          <measuredUnit>ms</measuredUnit>
75        </model.Operation>
76        <model.Operation>
77          <values>
78            <float>2722.0</float>
79            <float>4284.0</float>
80            <float>5805.0</float>
81            <float>5469.0</float>
82            <float>4053.0</float>
83            <float>2964.0</float>
84            <float>5006.0</float>
85            <float>1279.0</float>
86            <float>3264.0</float>
87            <float>2690.0</float>
88            <float>1227.0</float>
89            <float>1198.0</float>
90            <float>3995.0</float>
91            <float>5440.0</float>
92            <float>4517.0</float>
93            <float>2879.0</float>
94            <float>3689.0</float>
95            <float>4131.0</float>
96            <float>1672.0</float>
97            <float>4785.0</float>
98            <float>1309.0</float>
99            <float>5594.0</float>
100           <float>4731.0</float>
101           <float>4260.0</float>
102           <float>4102.0</float>
103           <float>4437.0</float>
104           <float>4605.0</float>
105           <float>2853.0</float>
106           <float>5294.0</float>
107           <float>5719.0</float>
108         </values>
109         <operationName>update</operationName>
110         <measuredUnit>ms</measuredUnit>
111       </model.Operation>
112       <model.Operation>
113         <values>
114           <float>3878.0</float>
115           <float>3262.0</float>
```

```
116             <float>1256.0</float>
117             <float>4955.0</float>
118             <float>4050.0</float>
119             <float>4639.0</float>
120             <float>3810.0</float>
121             <float>2400.0</float>
122             <float>3328.0</float>
123             <float>2849.0</float>
124             <float>5828.0</float>
125             <float>1537.0</float>
126             <float>5918.0</float>
127             <float>3985.0</float>
128             <float>3374.0</float>
129             <float>4737.0</float>
130             <float>4868.0</float>
131             <float>3364.0</float>
132             <float>4197.0</float>
133             <float>2183.0</float>
134             <float>4944.0</float>
135             <float>1749.0</float>
136             <float>3462.0</float>
137             <float>5506.0</float>
138             <float>2386.0</float>
139             <float>5559.0</float>
140             <float>2127.0</float>
141             <float>2410.0</float>
142             <float>5772.0</float>
143             <float>3230.0</float>
144         </values>
145         <operationName>retrieve</operationName>
146         <measuredUnit>ms</measuredUnit>
147       </model.Operation>
148     </operations>
149     <apiName>Alibaba</apiName>
150 </model.Evaluation>
```

APÊNDICE C – QUESTIONÁRIO

Abaixo segue as perguntas que componham o questionário. O mesmo pode ser acessado através do link: <http://goo.gl/forms/CyNt7QTFoM>. Com relação as perguntas que começam com "Em escala", o desenvolvedor seleciona de 1 à 5, onde 1 o item mais fraco e 5 o mais forte. A maioria das perguntas era caixa-seleção, mas o questionário contava também com perguntas abertas.

1. Qual é sua formação?
2. Em qual estado você mora?
3. Você atua no mercado de trabalho?
4. Quais das linguagens de programação abaixo você utiliza?
5. Há quanto tempo?
6. Você já programou ou programa aplicações baseadas em ontologias?
7. Você já precisou avaliar uma API, tais como manipulação de XML, persistência, dentre outras?
8. Com que frequência você precisar tomar uma decisão de escolha entre APIs?
9. Quais são os principais aspectos que você leva em consideração na escolha dentre as diversas APIs?
10. Em escala, o quão você sente a importância de um bom desempenho com relação ao tempo de resposta das operações oferecidas pelas APIs?
11. Em escala, o quão você sente a importância de um bom desempenho com relação à utilização de recursos (CPU e memória) pelas APIs?
12. Selecione, dentre as opções abaixo, quais as formas que você avalia a escalabilidade de uma API.
13. Em escala, o quanto você sente necessidade de um mecanismo que facilitasse a avaliação de uma determinada API?
14. Selecione dentre as opções abaixo, quais formas de visualização de dados seriam interessantes para você, na hora de visualizar os resultados de uma avaliação de uma API.
15. Em escala, o quanto você sente necessidade de outras formas de visualizações de resultados, além da textual, quando você está avaliando uma API?

16. Quando você está na dúvida dentre uma ou mais APIs, quais mecanismos você costuma utilizar para decidir qual API utilizar?

17. Em escala, o quanto você sente necessidade de um mecanismo que te ajudasse a comparar os resultados de suas avaliações das APIs?